USAISEC
US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

(1)

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

AD-A249 033

DTIC
ELECTE
APR 09 1992
S D
D

MULTIMEDIA NETWORK DESIGN

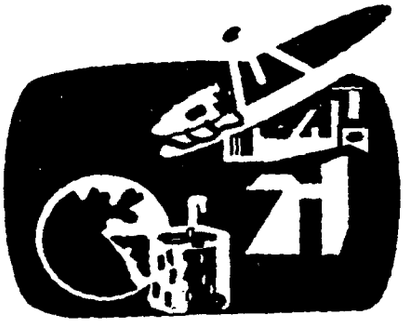
STUDY

(ASQB-GC-~~89~~ 90-002)

30 September 1989

This document has been approved for public release and sale; its distribution is unlimited.

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800



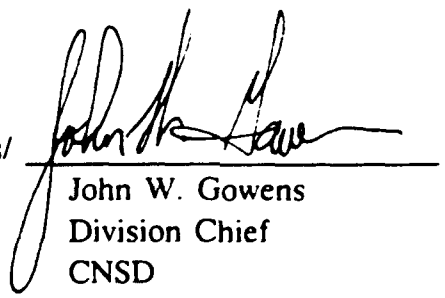
92-09106

92 4 08 106

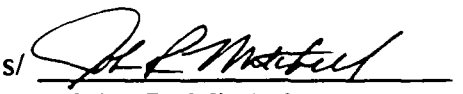
This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study with the work being done by the Harris Corporation. Its goal was to create a closed-form analytical queuing model for networks of queues. The Army's worldwide communication system has become a conglomeration of many systems such as DDN, DSN, AUTOVON, and wide or local area networks. The need for an efficient interconnection of these systems requires that systems be evaluated as a group as opposed to individual nodes. Therefore this research will provide a formal mathematical model specifically developed for the analysis of multimedia (e.g. coaxial cable, fiber optics, and twisted pair) networks. The primary output of the project is a PC/AT hosted program, MMDESIGN, which implements the formal mathematical model and provides a user interface for analyzing multi-media networks. The program is designed as an iterative analysis tool and can be used to derive steady state conditions for a network before attempting simulation analysis of the network to perform transient analysis. Pascal source code is provided in the report.

This research was performed under contract number DAK11-88-C-0052 for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ 

John W. Gowens
Division Chief
CNSD

s/ 

John R. Mitchell
Director
AIRMICS

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICAION AUTHORITY N/A	3. DISTRIBUTION/AVAILIBILTY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CLIN 0001 AA, SUN 0001 AB, CDRL A001/3	5. MONITORING ORGANIZATION REPORT NUMBER(S) ASQB-GC-002
6a. NAME OF PERFORMING ORGANIZATION Harris Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AIRMICS
6c. ADDRESS (City, State, and Zip Code) Melbourne, FL, 32902	7b. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800
8b. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS 8b. OFFICE SYMBOL (If applicable) ASQB-G	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAKF11-88-C-0052
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. P612783 / PROJECT NO. DY10 / TASK NO. 03-02-09 / WORK UNIT ACCESSION NO. DA 315 378

11. TITLE (Include Security Classification)

Multimedia Network Design Study: First Year Final Report

12. PERSONAL AUTHOR(S)

Dr. John R. Doner

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Annual	FROM OCT 88 TO OCT 89	89, SEP, 30	76

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUBGROUP	Multimedia, Network, Queing, Modeling

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study. Briefly, the study goals for the three years of the effort are as follows. In the first year of the study, now completed, the goal was to create a closed-form analytical queuing model for networks of queues. The second year of the study, now beginning, will build on the effort of the first year by enhancing the utility of the network-of-queues model to provide automated optimization capabilities. The third year of the study will attempt to integrate the use of this tool with a quantitative approach to define type mission of a communications system, and evaluating in an exact way the effects of the communication system on mission-oriented (not communications-oriented) metrics.

In the first year, a careful literature search was completed to determine the scope and depth of the selected modeling technique and its applicability to large-scale complex communications networks. The general result of this search was that the techniques of closed-form analysis are applicable to certain important areas of network design, and in fact complement, rather than replace, simulation techniques. The issues which can be dealt with by closed-form analysis relate to the proportioning of traffic across the available media in the system.

20. DISTRIBUTIONN/AVAILIBILTY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
CPT Timothy M. O'Hara	404/894-3136	ASQB-G

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

MULTIMEDIA NETWORK DESIGN STUDY

FIRST YEAR FINAL REPORT

SUBMITTED BY
DR. JOHN R. DONER, PRINCIPAL INVESTIGATOR

HARRIS CORPORATION
GOVERNMENT COMMUNICATIONS SYSTEMS DIVISION
P.O. BOX 91000
MELBOURNE, FL 32902

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification .....
By .....
Distribution /
Availability Codes
Dist | Avail and/or Special
A-1

TABLE OF CONTENTS

1.0	EXECUTIVE SUMMARY	2
2.0	THE RATIONALE FOR CLOSED-FORM NETWORK QUEUEING ANALYSIS	4
2.1	Closed-Form Modeling as an Adjunct to Simulation	4
2.2	An Explanation of the Closed-Form Network-of-Queues Model	7
2.3	Computational Considerations for the Closed-Form Technique	14
2.3.1	The Computational Process in a closed Network	15
2.3.2	The Computational Process in an Open Network	16
3.0	CLOSED-FORM MODELING APPLIED TO MULTIMEDIA COMMUNICATION	17
3.1	The Multimedia Network Node	17
3.2	The Multimedia Network Composite Channel	18
3.3	The Error Process for the Composite Channel	19
3.4	Accounting fot Error Correction Traffic	20
3.4.1	Erroneous Traffic Effects at the Transmitting Node	21
3.4.2	Erroneous Traffic Effects at the Receiving Node	21
3.5	Computing Transmission Delays through the Network	22
4.0	INSTRUCTION MANUAL FOR THE MMDESIGN PROGRAM	24
4.1	Explanation of Program Inputs	24
4.2	Program Organization and Menus	27
4.2.1	User Interface Format	27
4.2.2	Intepretation of the Menus	28
4.2.2.1	The "N(ew" Command	28
4.2.2.2	The "C(reate" Command	29
4.2.2.3	The "E(dit" Command	30
4.2.2.4	The "H(ardcopy" Command	31
4.2.2.5	The "R(ecall" Command	32
4.2.2.6	The "T(hruput" Command	32
4.2.2.7	The "P(aths" Command	33
4.2.2.8	The "M(etrics" Command	34
4.2.2.9	The "Q(uit" Command	35
4.3	Summary and Further Directions	36
5.0	REFERENCES	37
	APPENDIX A -- MMDESIGN MENU NAVIGATION	38
	APPENDIX B -- OPEN NETWORK MATHEMATICS	39
	APPENDIX C -- MMDESIGN SOURCE CODE	43

1.0 EXECUTIVE SUMMARY

This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study. Briefly, the study goals for the three years of the effort are as follows. In the first year of the study, now completed, the goal was to create a closed-form analytical queueing model for communication networks. The second year of the study, now beginning, will build on the effort of the first year by enhancing the utility of the network-of-queues model to provide automated optimization capabilities. The third year of the study will attempt to integrate the use of this tool with a quantitative approach to defining the mission of a communications system, and evaluating in an exact way the effects of the communications system on mission-oriented (not communications-oriented) metrics.

In the first year, a careful literature search was completed to determine the scope and depth of the selected modeling technique and its applicability to large-scale complex communications networks. The general result of this search was that the techniques of closed-form analysis are applicable to certain important areas of network design, and in fact complement, rather than replace, simulation techniques. Closed-form analysis of networks can only deal with steady-state equilibrium conditions in networks, such as the expected loading and delays in a network for which offered traffic, topology, and capacities have been allocated. This is quite different from the function that simulation performs, which is to study the consequences of specific scenarios in a network.

However, although simulation can yield very highly resolved insights into network behavior, it does so on a rather *ad hoc* basis: the simulator can test at most a very few of all possible communications scenarios which a network might be called upon to support. Closed-form analysis can provide globally applicable facts about a network, which may lead to early recognition of misallocation of capacity or potential for chronic overload. Using closed-form techniques, the network analyst can examine a wide range of network topologies and gain general insight into their suitability to meet mission requirements given the expected geographic and temporal variations in traffic load. These analyses will be much more rapidly executed than simulations, and the analyst can fairly rapidly determine which of a few candidate architectures appear in these general terms to support system requirements. Simulation studies can then proceed on this subset of architectures with the assurance that the candidate networks being simulated are at least close to the best answer satisfying system requirements.

Thus a well-executed design program for a large communications network should involve the interaction of simulation and closed-form analysis, with closed form analysis being used for a global estimation of the correctness of the network resource allocation, and simulation then following to test more specific aspects of the network design, such as routing strategies or survivability strategies. Such a design strategy will produce a more certain, and a less expensive answer, than will be obtained by simulation alone.

The intent of this study is to apply closed-form analysis to multimedia networks. A multimedia network will carry multiple types of traffic on multiple media. Each traffic type will be more or less suited for transmission on any given medium, depending on the medium bandwidth and error properties. Each traffic type may have certain essential constraints on its handling, related to timeliness and maximum permissible error acceptable for the traffic type. In a military

network designed for high survivability and maximum efficiency, the proper multiplexing of traffic types on the media can be an important factor in achieving such goals.

This requirement to multiplex traffic types across various media could be accomplished in several ways, such as allocating a specific proportion of each medium to each traffic type. The extent to which each traffic type would meet its timeliness and accuracy constraints would then be a function of that allocation. Since not every traffic type can at all times travel by the medium that is "best" for it, a process of compromise is necessary. It is precisely the determination of such an "optimum" compromise that is well served by the tools of closed-form network modelling.

The body of this document provides a formal mathematical model of multimedia traffic flow which encompasses the concepts of multiple traffic types, and multiple media types. The interaction of channel error processes with the traffic types is accurately captured, so that the multiplexing of traffic types on media types can be usefully analyzed. The primary output of this first year of effort is a computer program, called the MMDESIGN program which addresses the above analysis concerns. This program prompts the network analyst for a network design (i.e., topology, media, traffic types, routing, etc.), and then makes available the expected path delays associated with any traffic type over any path, or collection of paths. The program is designed as an iterative analysis tool: that is, the manner in which data is gathered, stored, and edited facilitates the analyst's normal activities in pursuit of the optimization of network performance relative to traffic multiplexing concerns.

The MMDESIGN program is hosted on an IBM PC/AT (or equivalent) and is written in Turbo Pascal, which is very widely available and well known to IBM PC programmers. This choice of computer limits the size of the network that can be handled, due to memory constraints, but the computer serves as a good base for wide dissemination of the program. The code is largely machine independent, and so could easily be ported onto a larger machine.

The remaining sections of this document provide an explanation of the closed-form modeling paradigm, its application to multimedia networks, and its implementation in the MMDESIGN program. Section 2 provides a rationale for and a description of the closed-form network-of-queues modeling paradigm. Section 3 explains the manner in which multimedia communications can be cast into that mold. Section 4 is a self-contained manual for the use of the MMDESIGN program. The appendices to the document provide complete detail and documentation of for both the mathematics and the computer code used in the program.

MMDESIGN will be extended in the second year to consider inclusion of optimization techniques within the program, such as optimization of capacity assignment and routing. The design of truly integrated multimedia communications systems is in a practical sense still in its infancy. It is hoped that this study will provide valuable tools to the operational network design community which must actually come to grips with the next generation of communications systems.

2.0 THE RATIONALE FOR CLOSED-FORM NETWORK QUEUEING ANALYSIS

In this section, we will introduce the rationale for the closed-form communications network modeling paradigm which has been the subject of this study. We will also provide a rather comprehensive survey of the applicability of the closed-form technique. This survey is not intended to be mathematically detailed, but does introduce terminology and concepts for the purpose of providing the reader with a comprehension of the general strengths and weaknesses of the technique.

2.1 Closed-Form Modeling as an Adjunct to Simulation

Modern military communications systems are rapidly evolving to take advantage of increasingly versatile communications technology. Procurement planning for the near-term future calls for increasingly survivable communications architectures which rely on an eclectic suite of communications assets. A major interest of all the military services is to fully integrate the use of multiple communications media into a single communications capability, the operation of which requires as little user management and intervention as possible. Such a system is expected to autonomously determine and ameliorate conditions detrimental to the expeditious flow of information, thereby creating a whole that functions better than the sum of its parts.

This idealized concept requires considerable innovation and experiment in the discipline of network control. Systems will in general comprise larger collections of assets, deal with a greater variety of traffic types, and be expected to handle larger volumes of traffic. Such designs will tax not only the traditional communications network design methods, but also the existing network design tools by which such designs are refined from concept to implementation.

The present study is a three-year effort funded by USA AIRMICS to consider the emerging design problems discussed immediately above. The intent of this study is to consider several concepts related to the design tools available to the network design community, and to create tools complementary to those now existing which will be specifically helpful in addressing the new multi-media, large-scale network designs of the near future.

This document reports on the results of the first year's effort, the establishment of a closed-form network-of-queues approach to modeling communications systems. Since most communications system design efforts rely heavily on simulation of the network, the rationale for creating a closed-form analytical model of network communications needs explanation. Simulation is, in essence, a process of determining single-point estimates of a very complex function. The inputs to the simulation constitute the independent "variable" (normally a multi-dimensional vector) of the function, and the outputs from the simulation constitute the value (again, normally a multi-dimensional vector) of the function. The user of the simulation selects an input value, runs the simulation, and obtains an output value.

Based on an iterative sequence of such simulation runs, together with modification of the simulation and/or its parameters, many major aspects of the network design may be determined. However, such simulation efforts generally constitute a sort of intuitive optimization process where the output of each simulation step guides the designer toward changes in the network design which will (it is hoped) provide better performance in the next simulation. In effect, the simulation user is attempting to discover the shape of a surface in a many-dimensional space by examining a sequence of "points" on that surface, and then selecting another value for the input argument to the function (i.e., simulation), which will move the output "point" uphill. This process is somewhat like that pictured in Figure 2 - 1 below. Since the simulator can only guess at the shape of the surface near the set of "points" already collected, it is never possible to assure that a network design based on simulation has actually achieved optimum performance within the design constraints.

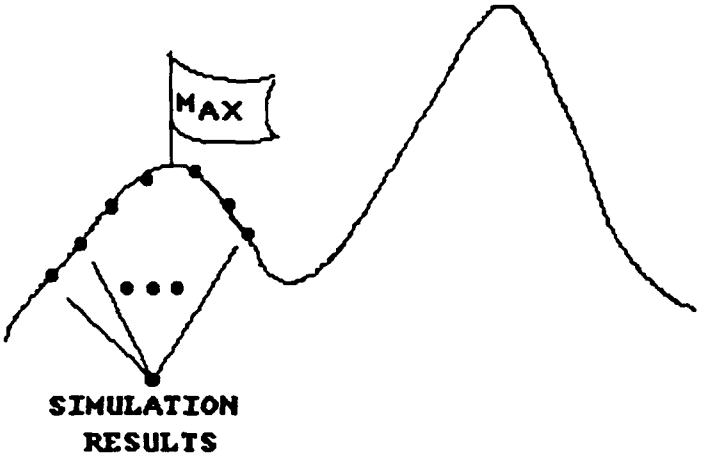


FIGURE 2 - 1: Simulation "Mountaineering"

Of course, the simulator, by examining as many "points" as possible on the simulation surface, can reduce the likelihood that there might be a better solution "near" the final chosen network design. But simulation as applied to modern and future military network designs tends to be expensive, and the number of events and communications paths to be simulated tends to grow combinatorially with the number of network nodes. As future networks move toward integration of all available media, it will be impossible to decompose the systems into multiple small networks, and so the need to handle very large numbers of events and assets in a single simulation will grow.

Having thus examined the conceptual and practical limitations of simulation, in what way can a closed-form, network-of-queues model contribute to network design? First, it should be admitted that such closed-form models are almost always bypassed in network design studies in favor of simulation. The reason for this is that a closed-form model can only model the functioning of a network operating in steady-state performance. Clearly, designers of military communications systems are very interested in guaging the response of the

network to many types of transient effects, and so cannot rely solely on steady-state network performance to select the parameters of their system. However, when only simulation is used, the steady-state performance can only be determined by long simulation runs, and each small change of input conditions may require another simulation run to obtain the changed steady-state.

Steady-state quantification within a closed-form network-of-queues paradigm is a much more convenient process. It is safe to say that a network designer could determine a great number of steady-state network solutions in the time required to determine a single steady-state solution by simulation. Moreover, since the closed-form model is analytical (i.e., expressed in terms of equations) in nature, there is the possibility of applying optimization procedures to the equations that describe the model, thereby obtaining a network design optimized in some respects directly from a single computer run of the model. Furthermore, this result may be a true optimum, rather than just a local maximum, as is more likely to happen when the optimization process proceeds essentially intuitively by means of simulation.

It is precisely this observation that justifies the use of a closed-form steady-state model of the system not in lieu of, but as an important adjunct to simulation. The designer then has an appropriate tool (simulation) by which to study system transient response, but can also more accurately "size" the network in terms of total assets required to meet the traffic demand of the system. An appropriate network design trajectory then uses the closed-form model to gain a global understanding of the network topology and link capacities required to efficiently meet the overall capacity demand at all points in the system. From this overview, simulation effort commences to resolve the more specific concerns of protocol development and allocation of resources at the individual nodes.

This interplay of simulation and closed-form analysis can also be used to advantage at later stages of system analysis. When network performance is to be analyzed over a range of scenarios including hostile actions against the system, simulations are usually done to demonstrate the manner in which the system recovers from loss of assets. Again, simulation is used here to examine system transient response as it moves from one steady-state (i.e., fully capable) to another. However, as was the case for design of the full network, if simulation is the only tool applied to this situation , then not all available information is being used. I.e., if the network transits from a fully capable state to an impaired state, then each of these states, through appropriate allocation of assets, can achieve some optimal performance relative to the network mission. If the optimum configuration in both circumstances is known, then simulation effort can be directed at fine-tuning network algorithms so as to obtain the transient response which moves the system toward its new steady-state in the most effective manner.

Summarizing the above points, simulation by itself is usually not adequate for a determination of the true optima possible for a network. As communications networks come to include ever larger suites of equipment, all integrated to serve as a single system, simulation alone will become less able to determine the best use of all the system assets, and the use of closed-form network-of-queues modeling will provide a valuable adjunct to the simulation effort. It does not provide the ability to examine specific protocols and routing techniques, as

simulation does, but it does permit the possibility of better global optimization and distribution of assets. A large-scale network design effort will generally be better served by using closed-form and simulation techniques together.

2.2 An Explanation of the Closed-Form Network-of-Queues Model

The terminology "closed-form" usually refers to technical results expressed in an equational format, such that all input parameters are independent variables of the equations, and the desired outputs are obtained by direct solution of the equations. When the technology in question is too complex to admit of such representations, it is usually necessary to rely on some sort of iterative solution procedure based directly on a mechanical characterization of the system interdependencies and how they serve to dynamically alter the system state. Solutions of problems in finite element analysis, in iterated differential (or difference) equations, and in simulation of system interactions all represent this genre of problem solution.

As was stated in the last section, most iterative solution techniques provide answers where no closed-form technique is available. Closed-form techniques, when available, have the intrinsic advantage of permitting mathematical manipulation and analysis of the equations involved, thereby providing the application of the great range of powerful mathematical optimization techniques available in the rich literature of optimization theory (see, e.g., [1] and [2]).

In the specific technology of queueing theory, the usual situation is that closed-form queueing techniques are confined to the characterization of single queues, or perhaps parallel queues with parallel servers all operating at a single service location. Most elementary queueing theory texts limit the development of the subject to such situations, and do not endeavor to discuss networks of queues at all. However, there is an extensive literature on this subject which has been evolving for about three decades, and has only recently found its way into textbooks and large-scale applications

Some of the earliest work toward extending queueing theory to networks of queues was done by R.R.P. Jackson (see [3]) in 1954. The main supposition which allows queueing effects at one node to be visited in an analytically tractable way upon the activities at other nodes is the assumption that the future behavior of the system as a whole is dependent on past behavior only in terms of the current customer backlogs in the system nodes. Making this assumption tended to place certain limits on the variety of queueing protocols which could be modeled, and some of these limits will be discussed below. Substantial progress was made after these early papers by various scholars, and the type of network in which the future of the entire network system could be determined solely from the present state of the queues at all nodes became known as "product-form" networks. A very significant paper in this area was published by four authors (see [4]), Baskett, Chandy, Muntz, and Palacios. The paper pulled together some of the disparate results in the area, and also extended the product-form queueing model to a wide and coherently described set of conditions. Lemoine gives an excellent survey of the technology in [5], and Lavenberg discusses the practical computational aspects of the technique in Chapter 3 of his excellent textbook, *Computer Performance Modeling Handbook*, [6].

The exposition to be given in this section will follow the form, but not necessarily the notation, of the BCMP paper. Also, the exposition in this section will try to provide the reader with a sense of the scope to which the product-form network theory can be applied, while leaving explicit mathematical development to Appendix B.

It will, however, be necessary to introduce some symbolic notation. First, suppose that

N = number of service centers (nodes) in the system, and

R = number of classes of traffic.

These classes of traffic are distinct from each other in that they can follow distinct routing schemes, and have distinct service time and arrival rate distributions at the nodes. Routing is defined probabilistically in such a network by

P[(i,r), (j,s)] = probability that traffic of class r, at node i will transit to traffic of class s, node j.

These are called routing transition probabilities, and are normally considered to be expressed as an NRxNR matrix which has great convenience for computational purposes. There is a simplification of notation possible here which does not affect the applicability of the above equation, and that is to regard customers of the same class at different nodes as being designated by different class indices. This evades the need to consider a matrix indexed by pairs, so we can then write the matrix P[ ] as

P[i, j] = probability that a customer of class i will transit to class j. (Equation 2 - 1)

Thus, routing permits traffic to move between traffic classes and service nodes in a single transition. However, the fact that routing is probabilistic means that no particular traffic entity travels any specific route through the system: the routing paradigm permits statements about average channel utilization and expected traffic flows along links to be made, but does not support a completely detailed routing plan. This is the reason that closed-form models cannot replace simulation for the purpose of examining the detailed effects of routing algorithms.

In most queueing situations such as this, certain traffic classes travel in closed routing chains. I.e., not all possible transitions between traffic classes may occur, and not all traffic classes visit all nodes. In typical mathematical fashion, the analyst can seize upon this opportunity to study the entire problem as a sequence of sub-problems. Thus, without too careful a formal exposition, we define a routing chain as consisting of a subset of traffic classes and a subset of nodes such that the traffic classes only transit among themselves, and the traffic classes involved only circulate among the nodes in the subset. This does not say that other traffic classes do not also pass through these nodes: also, if multiple routing chains do pass through the same queues, the overall state of that queue can be expressed from the analysis of the separate routing chains. In this way, the

analysis of the entire system can be done by analyzing the separate routing chains, and then extending these results to the interactions of the routing chains in order to assess the complete network system. Thus we will first describe the terminology and results associated with a single routing chain.

A routing chain is called closed if the total count of customers in the chain remains constant over time. Where this is not the case, the routing chain is open. Closed systems are often used to model the processing interactions and delays in a computer time-sharing system, where some constant number of tasks are being "simultaneously" served by several types of system servers (e.g., disk access, printer access, terminal access, CPU access), and the same number of jobs shuttle from one service to another. Computer system designers can judge the expectations of processing delay and resource utilization for a steady-state load of a given constant number of on-line time-shared jobs by formulating such a closed network of queues. Open routing chains may have arriving and departing customers, and thus one does not *a priori* know what total number of customers will be in the system at any moment.

We have progressed far enough now to state the most important computational advantage of product-form networks. The "state" of a product form network at any moment is (by earlier assumptions) given entirely in terms of the lengths and compositions (in terms of numbers of customers of each customer type) of the queues at the various nodes. I.e., the basic tenet upon which the product-form of network analysis rests is that the future of the network depends only on the present condition of the queues at all the service centers. Thus, the state of the network is equivalent to the collective states of the nodes, and the state of any node is entirely determined by the numbers of each class of customer in the queue of that node. Thus, if S_p is an R-dimensional vector the components of which represent the numbers of customers of each type in queue at node p, then

$$(S_1, S_2, \ldots, S_N)$$

represents the state of the entire network. If Pr{.} represents the probability of an event described within the brackets, then we may state that

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\} \qquad \text{(Equation 2 - 2).}$$

This equation states that the probability of the global state of the system, as represented by all of the individual queue vectors S_i is equal to the product of the probabilities of the respective queueing situations arising individually at the separate nodes. I.e., there are no intermodal effects and dependencies to affect the analysis, and we may divide and conquer the analysis problem by focusing on the behavior of a single node. The main result of the analysis is to provide closed-form expressions for the values $Pr\{S_i\}$, from which nodal response time, link utilizations, path delays, and other standard queueing metrics can be derived.

Having reduced the problem to examining single nodes in a single routing chain, we now taxonomize the types of queueing disciplines that may be treated by this analysis. First, all arrivals to the network have the Poisson distribution, and separate traffic classes may have separate arrival rates. The arrival rate for a given traffic class is usually defined globally as a single Poisson process which is

then subdivided among the nodes by a constant probability distribution, but it is equivalent to consider separate Poisson arrival rates at the individual nodes which sum to the global rate (the former conceptualization is more practical in terms of the mathematics of the model). The Poisson arrival rate to the system for any customer class need not be constant: it can be an arbitrary function dependent upon the number of customers of that class already in the system.

Of course, in a closed system, all arrival rates are taken to be zero. In an open system, if there are arrivals, then there must also be departures; the departure process is normally formulated in terms of a single "sink" for each traffic type, with traffic of that type being routed to the sink from each node via a routing transition probability. (This is logistically equivalent to some means of allowing customers to leave the system at individual nodes.) Thus the departure of traffic from the system is easily encompassed in the routing transition probability structure described by equation 2 - 1.

The final major element of the model has to do with allowable queueing disciplines and service time distributions. Product form assumptions can be realized for the following four general types of queueing disciplines. (Some other queueing disciplines have been shown to yield product form networks, but only for specialized topologies: e.g., see [7].)

The first queueing protocol permitted is the very common first-in, first-out (FIFO) queue. This is the most commonly encountered queue, where customers are placed in the queue in the order of their arrival, and are served in order of their arrival. Thus a newly arrived customer waits behind all previously arrived customers, and is served only after all previously arrived customers have completed service. Such a queue is illustrated in Figure 2 - 2. If a service node has a FIFO queue, then all customers which pass through that node are subject to the same exponential service time distribution.

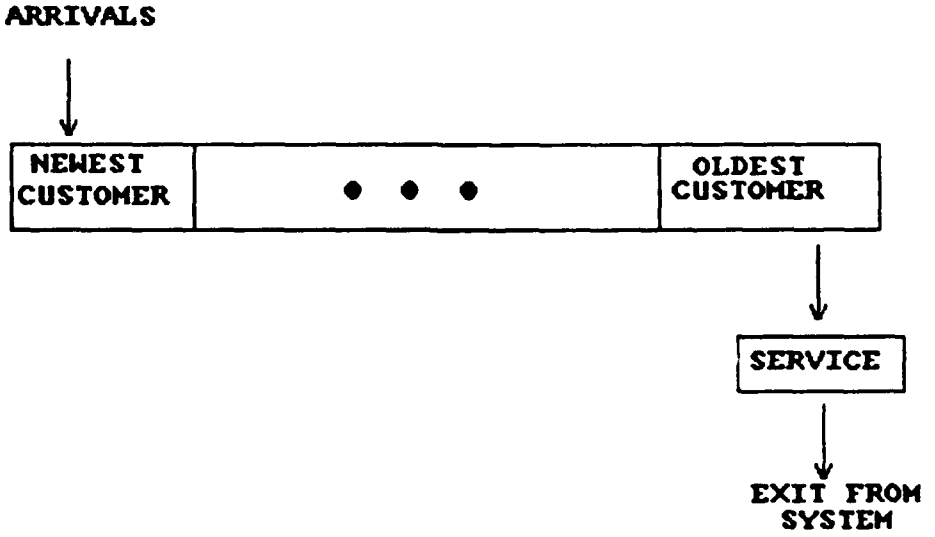


Figure 2 - 2 -- The First-In, First-Out (FIFO) Queue

The second type of queueing protocol possible at a service node is the processor-sharing (PS) mode of queueing. In this type of queueing, all customers at the node simultaneously share the server. Thus, each newly arriving customer receives immediate service, but the server accomplishes this instantaneous response only by slowing down the service rate at which all already present customers are served. Thus if the overall service rate is μ, then when K customers are present at the node each customer is served at rate μ/k. This type of queueing occurs in time-shared computer systems, and is illustrated in Figure 2 - 3.

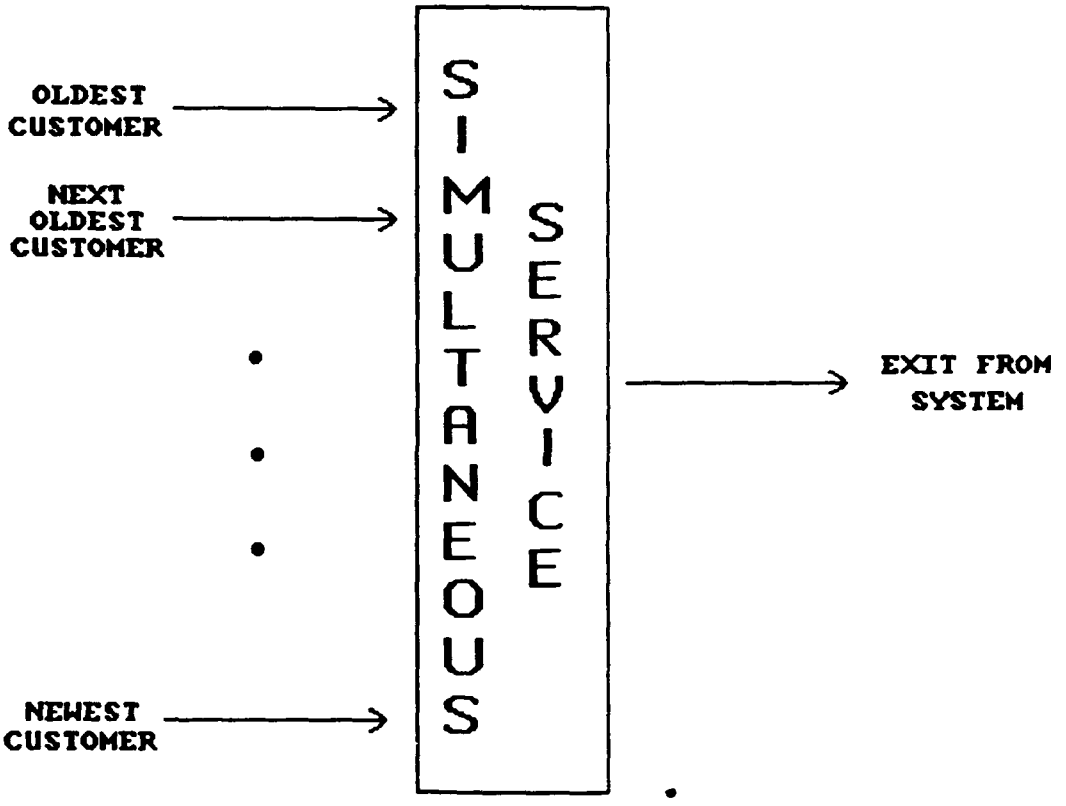


Figure 2 - 3: The Processor-Shared Queueing Discipline

Customers at a PS node can have distinct service rates, depending on their customer class. The service rate distribution can be any probability distribution with a rational Laplace transform. In effect, this means that the service operation can be thought of as consisting of a sequence of exponential service operations, each with independently determined mean, and with the possibility of the customer exiting the service after any one of the service steps. This type of service is depicted in Figure 2 - 4.

If a service operation of this type for customer class i contains only a single service operation, the service time distribution will be exponential. In this case, the only parameter of the service rate is μ_i, and $1/\mu_i$, the mean service time, can then be an arbitrary function of the number of customers of type i at the service center. Thus, μ_i can be expressed as a discrete function

$\mu_i(1), \mu_i(2), \ldots, \mu_i(j), \ldots,$

where the argument j represents the number of customers of type i at the node.

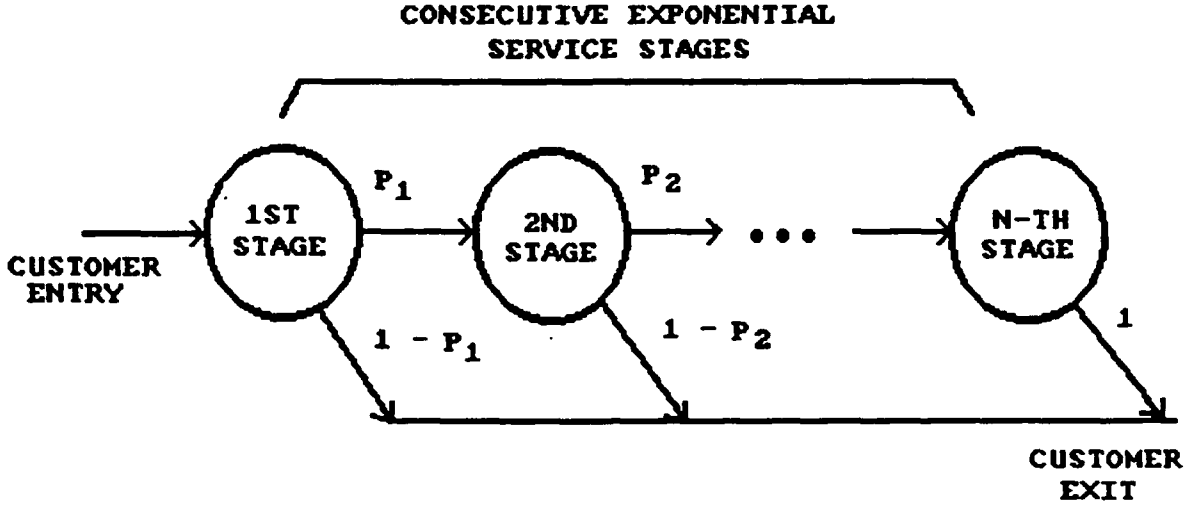


Figure 2 - 4 -- Schematic of Laplacian Distribution

The third type of queueing permitted at a service node is called infinite server (IS) queueing. In this case, the node always has more servers available than there are customers present in the node. The delay through such a node is thus strictly the service time delay associated with the customer class. The limitations on the service time distribution in this case are identical to those for the PS node described above. This node is illustrated in Figure 2 - 5.

IS nodes do not exist in real-world queueing systems, but they are useful when a single stage of delay is desired for a customer, with the delay being independent of other customer congestion in the system. E.g., in a communications system, a message which has waited behind other messages for access to a channel may, at the beginning of its actual transmission, wait for a channel access slot to become available in a round robin token-passing arrangement. This final short delay before transmission has nothing to do with other traffic in the system, and can be conveniently modeled by the IS queue.

The final form of queueing which is possible for product-form service nodes is last-come, first-served (LCFS) queueing. In this type of queueing, any newly arrived customer will preempt a customer already in service, and service for the preempted customer will then be suspended until the new customer has completed service. When a customer completes service, the most recently preempted customer resumes service. This type of service is evident, for example, in computer operating systems, where an interrupt to a processor causes the processor to suspend service to the present task and turn attention to servicing the interrupt. If another interrupt occurs during the service of the first

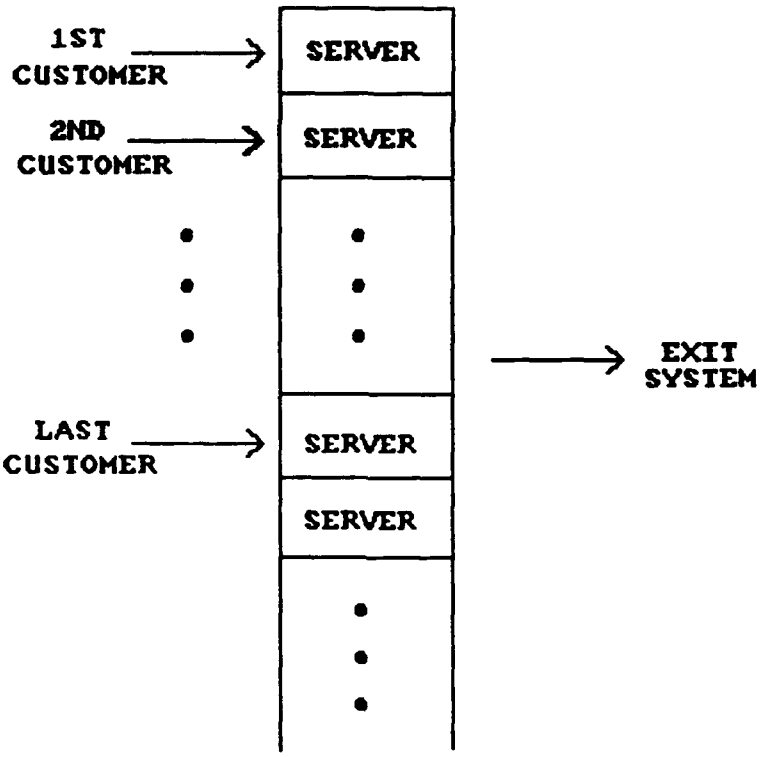


Figure 2 - 5: Infinite Server Queueing

interrupt, then the latter interrupt preempts the present interrupt. (Of course, many computer systems now have a prioritized interrupt structure, so that strict LCFS queueing would not apply.) LCFS queueing is illustrated Figure 2 -6.

The service time distributions possible for LCFS queueing are identical to those for PS and IS queueing. However, there is an added subtlety here, because preemptive queueing processes may or may not conserve the work already done on a customer. In the case of product-form LCFS queueing, the preemption is somewhere between conserving and non-conserving. Specifically, if the preempted customer has a multiple-stage service time distribution (see Figure 2 --3), then the customer is returned to service at the beginning of the stage in which preemption took place I e., the service already performed in earlier stages is preserved at preemption, but the service already performed at the current stage is lost. A final observation is that, for one-stage service time distributions, it follows that the preemption is non-conserving.

This effectively completes the description of the general topological and queueing models available for product form networks of queues. A given service center in such a network may apply any of the above four forms of queueing, and the flow of traffic, although stochastic, permits a customer of any class at any node to transit to any class and any node It should be pointed out that

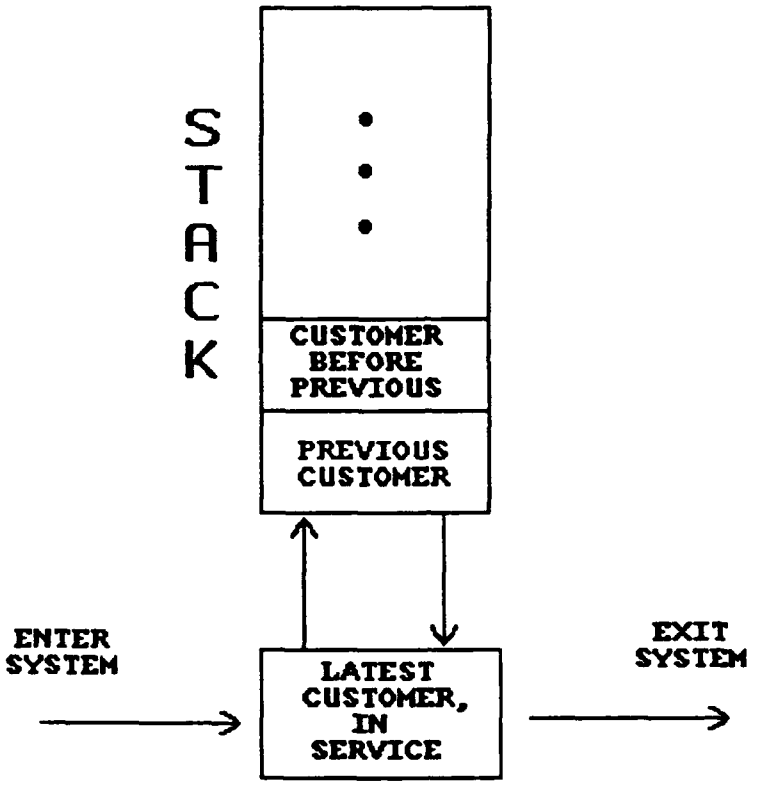


Figure 2 - 6: Last-Come, First-Served (LCFS) Queueing

customers can be effectively "deterministically" routed in this system by setting the appropriate transition probabilities equal to one. Also, it should be mentioned that the "service node" as described here is a mathematical artifact in terms of which the product form theory has been developed. In practice, the concept of a service node may involve several steps of processing of traffic in series and parallel combinations of the product-form service nodes. Thus, an actual communications network node may not be realistically reducible to a single node of one of the above types, but the delays and actions of the communications node may be adequately expressible in terms of a combination of the product-form nodes.

For example, suppose that we have a multimedia node at which messages of different types arrive. This node may be both processor-limited and bandwidth-limited, so that the nodal processing slows in proportion to the traffic in the node, and the traffic also backs up in queue at the output, waiting for service on the various media channels available . Such a node might best be modeled by a PS queue, followed by a FIFO queue.

2.3 Computational Considerations for the Closed-Form Technique

The present discussion would be incomplete without some reference to the practical computational complexity of applying the product-form network-of-

queues model. The difficulty associated with practical computation depends on whether the network is an open or closed network. The reason for this is that a fundamental quantity by which expected nodal loading, and thus all derived performance measures, are guaged is the "relative throughput" of each traffic class entering a node. The relative throughputs of a given customer class at a node is related to the routing in the system from all other nodes by the equation

$$y_d = P[0,d] + \sum_{c \in C} y_c P[c,d] \qquad \text{(Equation 2 - 3)}$$

where the y_d are the relative throughputs (one for each customer class/node pair in the network), and P[0,d] represents the arrivals to the class/node of new customers of the class d. The summation is taken over C, the set of all customer classes defined for the network. (The meaning of customer class follows the convention that each node/class combination is a distinct class, as was introduced in connection with Equation 2 - 1.) The situation now becomes quite different for open and closed networks, so these two situations will be treated separately in the following subsections.

2.3.1 The Computational Process in a Closed Network

The greatest computational difficulty arises from the fact that in a closed network, the quantities P[0,d] are all zero because there are no new arrivals to the system. The set of equations 2 - 3, with the quantities P[0,d] all set to zero, are linearly dependent because the coefficient matrix of the equations is Markovian and therefore has the sum of all columns equal to a vector of 1's. The result is that the relative throughputs, when solved for closed-form networks, are determined up to an unknown factor, i.e., the true class throughputs in the system constitute a vector which is a scalar multiple of the relative throughputs. The relationship is thus

$$(Y_1, Y_2, \ldots, Y_L) = (\alpha y_1, \alpha y_2, \ldots, \alpha y_L) \qquad \text{(Equation 2 - 4)}$$

where

Y_i = the absolute throughput for class i, and
α = the unknown constant relating absolute and relative throughputs.

The unknown value α is called the normalization constant. The process of determining α constitutes the bulk of the computational effort required to make the algorithm computationally feasible.

The unknown scalar α can be determined using the relative throughputs, by summing the values of the distribution

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\}$$

(see Equation 2 - 2) which theoretically must add to one. When the factors on the right are individually computed, using the relative throughputs, and the probability density in equation 2 - 2 is summed over all possible values of the nodal state vectors $S_1, S_2, \ldots, S_N$, the resulting sum will be in error by exactly the factor α.

Although straightforward enough in concept, this summation can be very large, since it effectively involves enumerating all possible combinations of queue backlogs jointly considered over all nodes. (However, since there are a fixed number of customers in a closed network, the computation is not infinite.)

A great many of the papers in this field have been devoted to decreasing the computational complexity associated with this step. Three main techniques are prominent, each of which may be favored under certain circumstances (see [6], pp. 145 - 151 for an excellent exposition of these techniques). The three techniques are known as recursion, mean value analysis, and the local balance algorithm for normalizing constants. Two very recent major algorithmic approaches to the computation of normalization constants are the RECAL algorithm described in [8] and the DAC algorithm discussed in [9].

2.3.2 Computational Process in an Open Network

The great difficulty involved in computing normalization constants for closed networks disappears completely for open networks. That is because in Equation 2 - 3, the arrival rates are non-zero, and so the linear equations are no longer singular. Consequently, the main computational difficulty is simply to solve the linear equations represented by Equation 2 - 3. This effectively can be done by any of a large number of efficient matrix inversion techniques. Generally, the major limitation of the technique for the open network is the size of the matrix of routing transition probabilities. Bear in mind, however, that where the customers in the system break into a number of disjoint routing chains, the full set of equations represented by Equation 2 - 3 also decomposes into smaller sets of disjoint independent equation sets. In the types of network applications that we will pursue, we will normally be dealing with open networks, decomposable into a number of disjoint routing chains (in fact, one chain for each traffic type). Therefore, the computational effort described in this section need not reflect the full complexity of the network in a single large set of linear equations.

3.0 CLOSED-FORM MODELING APPLIED TO MULTIMEDIA COMMUNICATION

The previous section of this document gave the reader a survey of the applicability of the closed-form, network-of-queues modeling techniques, and of the computational complexities involved. Based on that survey, the present section will examine the multimedia communications network, and provide a modeling paradigm for that network which utilizes product-form network-of-queues techniques.

Before launching into this development, it will be worthwhile to clearly state that the purpose of this modeling effort is to provide a means of studying the multiplexing of traffic types on the media types in a multimedia network in such a way that a fully integrated multimedia network (as opposed to a collection of separate single-medium networks sharing common nodes) results, with optimum use of the media relative to the characteristics of the traffic types.

A final point that should be mentioned is that the model developed here is in effect a prototype, and is kept fairly simple for that reason. It treats the network in a rather simplified form, and is limited in scope to the examination of traffic loading issues in the system. Since this is the first year's output for a three-year study, the prototype will be expanded in many ways to serve a more detailed set of issues in multimedia networks. To what extent this prototype can be expanded in scope will depend on further experience gained with the prototype and with the computational efficiency of the prototype. This can only be ascertained after the prototype has been exercised over some range of examples in the second year of the study.

This section includes some mathematical development which is essential to understanding how the closed-form modeling paradigm has been adapted to multimedia communications networks. The mathematics developed here is not part of the general mathematics of product-form networks of queues; it was developed explicitly to correctly represent multimedia communications concepts in terms of the product-form model. A user of the MMDESIGN program must understand the concepts in this section in order to intelligently apply the MMDESIGN program to design issues.

3.1 The Multimedia Network Node

A multimedia network node will be characterized for our purposes as a node which accepts traffic from other nodes, and on various input links, and can then pass traffic from itself to other nodes along various links. The links entering and leaving the node can be supported by various media and modulation types, and the traffic entering and leaving the node can be of different types. The main distinctions which will be drawn between media in this model will be the bandwidths which each medium makes available for traffic, and the signal degradation properties of the medium/modulation pair as it effects each traffic type's error rate. The main distinction between traffic types that will be drawn will be the differing error rates induced by the various media, and the mechanisms by which erroneous traffic is handled.

Internal to the network node, we must identify a queueing discipline which is compatible with those supported by the product-form model, and is also compatible with our desire to model traffic flow realistically in a communications system. Of the four queueing disciplines available (see Section 2.2), the FIFO discipline is the most reasonable match to ordinary traffic queueing. I.e., traffic leaving a node may need to be queued because the bandwidth available on some medium is less than required to immediately service the current traffic offered.

However, an irritating complication arises if we simply try to model each medium leaving a node as a single FIFO queue, and that is that the FIFO queueing discipline for product-form networks constrains all customers entering the queue to have the same service time distribution. This would be acceptable if a single traffic type were to be matched to each medium, but it will not do if we are to accurately reflect the transmission of multiple types of traffic on a single medium.

The remaining queueing disciplines allowable for product-form networks permit separate customer classes in the queue to have separate service time distributions. These three queueing disciplines however (i.e., processor sharing, infinite servers, and last-come, first-served) do not intuitively map well into our concept of traffic queueing at a node, and would not provide an applicable model.

The consequence of all this is that the FIFO queue should somehow be used, but should be limited to serving a single traffic type. Fortunately, this is possible to do in a credible way, and this will be the subject of the next section.

3.2 The Multimedia Network Composite Channel

As was mentioned immediately above, we cannot model separate media channels as separate queues within the constraints of the product-form model unless we attribute the same service time distribution to all traffic using that channel. This would seem to preclude the multiplexing of multiple traffic types (each of which may have its own distinct service time distribution) on a single channel. In order to circumvent this problem, we will instead regard a channel as carrying a single traffic type, and we will in effect multiplex the channels for the traffic type.

In order to explain this concept, we must introduce some notation. Given a single traffic type j at a specific node i, that traffic type is to be multiplexed on the various media available for transmission. Define a traffic/medium multiplexing vector as

$$M^{ij} = (m^{ij}_1, m^{ij}_2, \ldots, m^{ij}_T),$$

where

m^{ij}_k = the proportion of medium k to be devoted to traffic type j at node i, and

T = the number of media available at the node.

Traffic type j at node i will be apportioned as shown on the various available media. This means that the proportion m^{ij}_k of medium k is set set aside

exclusively for traffic of type j.

In effect, this defines a composite channel for traffic of type j. The bandwidth of the composite channel is expressible as the sum of the proportions of the bandwidths of the media channels partially supporting the traffic type. To be precise, if medium k has bandwidth B^i_k at the node i, then the total bandwidth allocated to traffic type j is

$$B^{ij} = \sum_{k=1}^{T} m^{ij}_k B^i_k \qquad \text{(Equation 3 - 1)}$$

The advantage of the composite channel concept is that it presents to the traffic type in question a total bandwidth available, as per the multiplexing scheme of the node, and it allows the representation of the separate traffic types as traveling on separate channels. In this way, all traffic entering a composite channel is of the same type, i.e., has a single service time distribution, and so the queueing discipline associated with the composite channel can be taken to be FIFO.

The composite channel must also be considered from the standpoint of error processes acting on traffic. This will be examined in the next section.

3.3 The Error Process for the Composite Channel

The composite channel comprises, for its related traffic type, a collection of fractions of media channels, each of which may have different error properties relative to the traffic type. The composite channel error rate is therefore dependent on the specific proportions of the various media available to the traffic type for which the composite channel is defined.

Before carrying this reasoning to a precise expression, it will be useful to quantify the error process somewhat more than it previously has been. For each medium/modulation combination, there is some form of signal degradation representing the usual operational characteristics of the medium so modulated. Whatever form this degradation takes, it will affect any specific traffic type to an extent depending on the error-correcting mechanisms built into that traffic type. For the purposes of the current model, we will assume that all traffic types can be thought of loosely as "messages" (the term "packet" seems too dangerously specific), and for each medium/traffic type combination, a missed message rate (MMR) will be determined based on a careful analysis of both the medium/modulation and the traffic type.

Thus for traffic type j and medium k, we will denote a missed message rate (MMR) by E_{jk}. The composite channel can be assumed to carry traffic in direct proportion to the band width allotted per medium type, so the composite error rate for traffic type j at node i should be expressed as

$$E_j = \sum_{k=1}^{T} m^{ij}_k E_{jk} \qquad \text{(Equation 3 - 2).}$$

This error rate is thus interpreted to be an overall missed message rate for the

composite channel. The average missed message rate for all traffic of this type traveling on the composite channel will correspond to this MMR.

3.4 Accounting for Error Correction Traffic

The subsection above dealt with determining the error rate for a composite channel relative to the traffic type that will flow on that channel. The error rate will be applied to determine how many messages (in the present circumstances, the term "messages" will be regarded as a generic term for separate traffic entities) transmitted on the composite channel will be received in unacceptable condition. When received traffic is unacceptable or unusable, there are generally three possible responses to the situation:

(1) the traffic is discarded, with no need for a repeated transmission,

(2) the traffic has substantial forward error correction built in, so the receiving node can resolve the problem with no further use of communications links

(3) the traffic must be retransmitted.

For the purposes of the present model, we are only concerned with processes that increase the burden of the available media. Therefore, we need only concern ourselves with error handling of the third kind. For such error handling processes, we shall assume that each message subject to error correction, when received correctly, is acknowledged by the receiving station. This acknowledgement will generally consist of a short message returned to the transmitting node using the same composite channel. If the received message is not correct, then no acknowledgement is sent.

There are several ways in which the mechanisms of such error handling could be modeled. In keeping with the philosophy of steady-state modeling, we must bear in mind that the purpose of this model is not to follow specific traffic entities through the system, but rather to guage overall traffic congestion and delay through the system. (Actually, since routing in the product-form model is not deterministic, there is no way to account on a message-by-message basis for erroneous traffic transmission.) Therefore, so long as the additional traffic load imposed by error correction is modeled, it is not necessary to actually implement flow paths representing the handling of acknowledgements and retransmissions.

What additional traffic loads are imposed by this error correction scheme? First, there is the additional load arising from the need to transmit repeats of erroneous traffic along the original path. Second, there is the need at the receiving end to generate and return acknowledgements to the transmitting station for correctly received traffic entities. We will account for all effects of erroneous messages by adding an extra load at the transmit node which is equivalent to the additional traffic it must transmit due to the error process. Since the extra load occupies the same FIFO queue as all normal traffic for the affected composite channel, the overall effect on the system is an additional amount of delay in the node due to the need to requeue and retransmit some fraction of the channel traffic. For the acknowledgement process, the extra load is imposed on

the receiving node, since it must use some of its transmit capacity to queue and transmit an acknowledgement.

To adequately account for these added traffic loads, it is necessary to analyze the intensity of traffic flow for the traffic type in question, and then use that intensity figure to calculate the extra traffic loading imposed on the queues in both the transmitting and receiving node. We will do this in the two subsections below.

3.4.1 Erroneous Traffic Effects at the Transmitting Node

If we are dealing with a specific traffic type t, the added load at the transmitting node is a function of its mean length l_t and the missed message rate E_t for the traffic type. In particular, for every original message transmitted, the total load at the transmit node is just

$$S'_t = l_t(1 - E_t) + 2l_t(1 - E_t)E_t + 3l_t(1 - E_t)E_t^2 + \ldots$$

This infinite summation does have a closed form for $0 < E_t < 1$, and yields

$$S'_t = l_t (1 - E_t)^{-1} \qquad \text{(Equation 3 - 3)}$$

In effect, this is the expected length of all traffic generated at this node associated with the original message. By lengthening the nominal traffic length l_t to this value, we have imposed the desired additional load on the node.

3.4.2 Erroneous Traffic Effects at the Receiving Node

We will handle the effects of acknowledgements on the traffic process by also increasing the length of each type t message handled to account for the acknowledgement it requires back to the previous node. However, not all traffic of type t which is in queue at the receiving node has actually been received from other nodes. I.e., the traffic in queue at the node is generally a mixture of received traffic, and traffic originated at the receiving node. Obviously, the node need not generate acknowledgements for traffic internally originated, therefore only some fraction of the type t traffic actually imposes a load on the node. Thus, in order to assess the load at the node correctly, we wish to determine the fraction of traffic of type t originated in the node, relative to all type t traffic processed by the node.

This is not actually a difficult thing to do, since we have available the relative throughputs from Equation 2 - 3 . In terms of the notation of Equation 2 - 3, suppose that P[0,d] represents the originations for traffic type t, and that y_d represents the relative throughput of traffic type t at the receiving node. Then the fraction of traffic which represents local originations of traffic type t, compared to all traffic of type t processed by the node is

$$v_t = P[0, d]/y_d \qquad \text{(Equation 3 - 4).}$$

Then the average length for all type t messages processed by the node is given

by

$$S_t = v_t [l_t (1 - E_t)^{-1}] + (1 - v_t)[l_t (1 - E_t)^{-1} + a_t] \qquad \text{(Equation 3 - 5)}$$

where

a_t = length of the required acknowledgement for traffic type t..

Thus, the overall additional load imposed by the error process is visited on the system effectively by increasing the length of each message to account for its retransmissions by the system, and the acknowledgements sent on its behalf. Thus given the quantities l_t , E_t , and a_t , we can calculate for each node (note that it is a function of each individual node's traffic type t throughput and origination rate) the length S_t for the traffic of type t at that node. This effectively determines the service rate for that traffic type at that node.

Suppose that we are dealing instead with the situation where acknowledgements are sent "out-of-band", i.e., on a channel other than the one which carries the original traffic. Then the additional load on the original traffic channel reverts back to the value S'_t. The remaining part of the traffic generated is the out-of-band component associated with the acknowledgement process, i.e., just the load associated with the generation and transmission of the correct proportion of acknowledgements for the traffic type. That will be given by the difference

$$S_t - S'_t .$$

3.5 Computing Transmission Delays through the Network

The final remaining topic which is to be considered in this section has to do with the means by which path delay through the network can be computed. For the open network product-form model, the computation of all nodal performance metrics is very straightforward once the linear equations (Equations 2 - 3) determining the relative throughputs have been solved. One of the nodal performance metrics available is the mean nodal response time (i.e., queueing delay plus service delay) for each traffic type passing through the node. (See Appendix B for the derivations of system peformance parameters from the relative throughputs.) In order to compute the expected delay for any traffic type along a path through the network, one can add the mean nodal response times for the nodes along that path.

However, if what is desired is an average delay for traffic over a multiplicity of paths, then one cannot simply take the unweighted average of the path delays described in the above paragraph. That is because one cannot assume that equal amounts of the traffic of interest flowed via the various paths. Thus, we must find a means to account for the relative proportion of traffic that flowed along any one path among a collection of paths of interest.

This can be done by reference back to the relative throughputs defined by Equation 2 - 3 . Specifically, let

$$P = < s_1, s_2, \ldots, s_{n(p)} >$$

represent a path through the network, with s_1 being the first node, s_2 being the second node, etc. The expected path delay along this path for traffic type t will be the sum of the expected response times for traffic type t at each node included in the path. Let

$D_t(P)$ = expected delay for traffic of type t traversing path P.

Then if we have another path Q connecting the same origin and destination, a constant α must be such that the expected delay for all type t traffic traveling between these endpoints on both paths can be expressed as

$$E[D_t(P \text{ or } Q)] = \alpha\, D_t(P) + (1 - \alpha) D_t(Q).$$

We determine the constant α from the relative throughputs of the nodes (for traffic type t) and the routing transition probabilities. Beginning at the next to the last node on path P, the proportion of all type t traffic through the node destined for node $s_{n(p)}$ is given by the routing transition probability $P_t[s_{n(p)} - 1, s_{n(p)}]$; multiplying this by the relative throughput $y_{t,n(p)} - 1$ of node $s_{n(p)} - 1$, i.e.,

$$P[s_{n(p)} - 1, s_{n(p)}]\, y_{t,n(p)} - 1,$$

gives a relative measure of the type t traffic traveling this link. Now, from this quantity of traffic, we wish to know what portion arrived at node $s_{n(p)} - 1$ from the preceding node, $s_{n(p)} - 2$. Applying the same reasoning to this situation, we determine a measure of the relative traffic of type t at node $s_{n(p)} - 1$ from node $s_{n(p)} - 2$ as being

$$P_t[s_{n(p)} - 2, s_{n(p)} - 1]\, y_{t,n(p)} - 2\ ,$$

where $y_{t,n(p)} - 2$ is the relative throughput of traffic type t at node $s_{n(p)} - 2$.

This reasoning can be extended inductively backward to the first node of the path, with a similarly defined factor applying at each node. Thus a measure of the relative amount of traffic flowing from node s_1 to node $s_{n(p)}$ is given by the product

$$W_t(P) = \prod_{i=1}^{n(p)-1} P_t[s_i, s_{i+1}]\, y_{t,i}\ . \qquad \text{(Equation 3 - 5)}$$

Thus if the expected delay for type t traffic is to be computed for travel along any of the multiple paths, say $P_1, \ldots, P_n$ it will take the form

$$E[D_t(P_1 \text{ or } \ldots \text{ or } P_n)] = \Big(\sum_{i=1}^{n} W_t(P_i) E[D_t(P_i)]\Big) \Big/ \Big(\sum_{i=1}^{n} W_t(P_i)\Big) \qquad \text{(Equation 3 - 6).}$$

This effectively completes the discussion of model development which was the main subject of this section. All of the technical results presented above are specific to this application, and are generally not part of the generic results derived in product-form network-of-queues expositions. Appendix B provides a full accounting of the generic mathematical treatment of the open product-form network which is sufficient for our purposes.

4.0 INSTRUCTION MANUAL FOR THE MMDESIGN PROGRAM

The main goal of this year's study effort was to develop the mathematics needed to create a credible model of the multimedia network within the product-form network-of-queues framework, and to then implement the concepts in a computer program. The MMDESIGN program developed for this purpose is effectively still a prototype, and will undergo considerable generalization and improvement in the next year. Therefore, the content of this section should not be taken as a permanent record of the capabilities, form, or user-interface associated with this program. Many of the features described in this section are still in development, and others are not yet fully debugged.

The specific objective of this program is to provide an analytical tool enabling communications network designers to assess the tradeoffs involved in assigning various traffic types to various media supporting a multimedia network structure. The tradeoffs relate to the greater or lesser ability of a given medium to service any particular traffic type within the constraints of traffic degradation and delay. Where some media are superior to others relative to these properties, some portion of the traffic may need be relagated to the poorer media. This program will aid analysts in determining the steady-state effects of traffic multiplexing on the media.

MMDESIGN in its present form does not perform any automated optimization of routing. The user can enter the information defining the network, and can then derive and examine the steady-state performance of the system. The primary metrics provided are the expected response times (i.e., queueing delay plus service time) for each node and traffic type, the expected delay times for any traffic type traveling any specific path or collection of paths all of which have the same origin and destination.

This program is quite data intensive, since it will require enough information to completely specify all routing in the network, all traffic types (each of which has its own routing structure), and all media Thus this program is not "user friendly", in the sense that one can get practical results from its use in short order. To fully specify a large network to the level required for this program could require substantial tedious data input. However, once that data has been supplied, it is possible to examine many traffic multiplexing scenarios with much less expenditure of time and much greater confidence in the results than would be available through simulation.

The following subsection will be devoted to explaining the meaning of each data input required to fully define a multimedia communications network to the program.

4.1 Explanation of Program Inputs

The modeling techniques described in Section 3 permit the network analysis done by MMDESIGN to be done individually by traffic type. That is because the channel multiplexing technique used to create a composite channel makes the

traffic types independent of eachother, except that each traffic type has a limited amount of the total media bandwidth available to it, associated with which is a composite traffic service rate and traffic error rate.

Because this is the case, the data entry process for MMDESIGN is organized primarily around traffic types and the specific information associated with a single traffic type. Furthermore, the information input scheme is such that analysis can proceed for a single traffic type once all of the data associated with that traffic type has been entered.

Since the amount of data to be input for an entire network can be very extensive, the program organization only keeps data for a single traffic type in computer memory at any time. This is of great advantage in the present IBM PC implementation, since most IBM PC's or equivalents have less than 1 megabyte of memory capacity.

Data input for any network design analysis is automatically stored to a file as it is entered. This file can then be invoked in a later session and used as is for further analysis, or edited if it is desired to try a different, but similar network configuration. There is one limitation built into the data storage retrieval process which was unavoidable, given the constraint on available memory, and that is that, although almost any of the originally entered data associated with a network can be edited, the overall "size" of the network must remain the same. In this case, the size of the network is a function of the number of nodes, the number of traffic types, and the number of media types input in the network definition. Once these three values are selected, a new network obtained by editing the present network cannot change any of them. (A larger network can be defined only by going through the full network creation process again.) Thus, if one defines a network, and anticipates that the network later may involve more traffic types, media, or nodes, one should select the maximum values expected for those numbers at the initial creation of the network. Doing so does not measurably add to the workload associated with data entry until such time as the network definition is actually expanded.

The inputs to the program during network creation will now be discussed, in the order in which they are input. First are the <u>global</u> parameters, so called because they are not associated with any single traffic type; these are

a. the total number of communications nodes in the network,

b. the total number of media types in the network,

c. the total bandwidth (in Kbits/second) for each medium type,

d. the total number of traffic types in the network.

Following the entries above, the block of data entries discussed below are all associated with a specific traffic type. The user enters these parameters in consecutive blocks of entries, with all entries in a given block being associated with a single traffic type. After the data for any traffic type has been entered, the user may exit the network creation process and proceed to the analysis portion of the program for the traffic types already defined. The traffic type data entry

comprises

a. the network-wide traffic arrival rate for the traffic type,

b. the mean message length for the traffic type,

c. the length of the acknowledgement for the traffic type (enter 0 if no acknowledgement is used)

d. a collection of missed message rates for this traffic type, one for each medium on which it will be transmitted,

e. a collection of local arrival rates, one for each communications node in the system (these rates correspond to the probability distribution by which the global arrivals for a traffic type are subdivided, as explained in Section 2.2),

f. station traffic multiplexing vectors by which the composite channel for the traffic type are defined (see Section 3.2), one multiplex vector being required for each node in the network,

g. the network routing transition probability matrix P[i, j] for the traffic type, i.e., the defined routing in the system may be varied by traffic type.

The quantity of data required to define a large network is extensive, especially since the items e. through g. must be entered for each traffic type, and some of those items (especially the station multiplex vectors and topology routing matrix) may require substantial numbers of individual entries. However, there is available in the program a copy feature that allows the most voluminous data structures, if identical for different input cases, to be copied from previously entered data. E.g., if, for a given traffic type, all station traffic multiplexing vectors are to be identical, then the copy function allows the analyst to evade a very substantial amount of data entry.

A final data entry process, which is decoupled from network creation/editing, is associated with the determination of the paths over which the model is to compute traffic delay. The inputs in this case specify to the program which network paths are of interest to the user in computing path delay through the network. The user may in effect enter sets of paths, and the final performance output for the program will compute the expected delay for each traffic type along those paths, with the delays computed being averages taken over each path set. The path data need not be entered at the time that the network data described above is entered. The user can enter network definition data, and compute all nodal metrics of interest, if so desired, before proceeding to evaluation of path delays. All path data is entered under a separate menu function "Paths", at a time of the user's choosing. The path data entered is

h. number of sets of paths to be defined,

i. a path description, entered as a sequence of nodes (interpreted as from origin to destination), and the path set in which it is to be included.

As is the case for all network definition data, the path data is stored to disk, and can be recalled and edited at will in association with the network data defining the network.

A final point concerning the total aggregate of input data is that it is possible to enter data in so complex a model as this which is mathematically inconsistent. There are four possible forms of inconsistency , namely,

1. the possibility that the row sums of any routing transition probability matrix do not equal 1 (the row sums are probability densities, and so must add to 1), associated with data entered under item g. above,

2. the possibility that the local arrival rates for a traffic type do not sum to one, associated with data entered under item e. above,

3. the possibility that more than all channel bandwidth for a given media type might be allocated to the various traffic types, associated with data entry under item f. above,.

4. the possibility that a defined path, as entered by the user in connection with item i. above , is not in fact supported by the media routing transition probability matrices for any of the traffic types.

There are "Verify" utilities provided in the program to assist the user in checking that each of the above data types is consistent. A verification function is automatically invoked at the end of a network creation session, to inform the user of any difficulties detected relative to items 1. to 3. above. That utility can also be invoked by the user after any network editing, in order to insure that previously consistent data has not been made inconsistent by the editing process.

Of course, there are other possibilities for what amounts to inconsistent data entry, such as entering parameters which are obviously out of range, or which create hopelessly large traffic loads in the network. There is no range checking for such data errors in the program.

4.2 Program Organization and Menus

This subsection will describe the user interface to the MMDESIGN program, and will explain each program function in detail. It is important to reiterate at this point that MMDESIGN is a prototype program, and will evolve substantially over the next year of this study. Therefore, the material in this manual concerning program interface and function is interim information.

4.2.1 User Interface Format

The MMDESIGN program is entirely menu-oriented. It consists of a main menu which requires single keystroke responses from the user, and several submenus associated with main menu commands. The general format of all menu lines is the same: each command on a menu line is written in the form

"X(xxxxxx ",

and the user must enter the first letter of the command (in either upper or lower case) , followed by the "ENTER" key. This results either in the presentation of a submenu or specific prompts for data entry associated with the command. In general, all possible user actions are met with clear prompts for the appropriate action.

The other major aspect of MMDESIGN screen format is the division of the screen into two portions. The top portion comprises about one-fifth of the total screen, and provides a status/navigation window to the user. This window displays at any time the current menu level at which the user is active (shown hierarchically from the top menu), as well as the name of the network file and the traffic type currently under investigation. If no network file has been opened, then the name displayed is "Undefined".

The lower portion of the screen is the user/program interaction screen, and effectively functions as an ordinary terminal interface, with data scrolling off the top when the screen becomes full, and new data is entered at the bottom.

The program contains, together with the verify functions, a number of other warnings indicating fatal problems, such as an inability to open a requested file. Warnings of this type are presented in blinking red text, and are preceded by a short tone from the computer speaker.

4.2.2 Interpretation of the Menus

In this section, each of the MMDESIGN commands available through the program menus will be explained. Since the menu structure is hierarchical, menus at lower levels may be referred to with simultaneous reference to their ancestors in the hierarchy, where that improves the clarity of the presentation. Such references will take the form "PARENT/CHILD/GRANDCHILD/...". In this notation, the main program menu will be referred to as "MAIN". While reading to the material below, the reader may find it helpful to refer to the template in Appendix A which provides a view of the full hierarchical menu structure of the MMDESIGN program.

The MAIN menu is presented as the following line:

" N(ew, C(reate, E(dit, P(rint, R(ecall, T(hruputs, P(aths, M(etrics, Q(uit: "

To facilitate document organization, the discussion will be broken into separate subsections below. It should be noted that a thorough reading of the following subsections is mandatory before attempting use of MMDESIGN, because many essential details of program operation are embedded in the following discussion, and may effect the understanding of commands other than those under which they are introduced.

4.2.2.1 The "N(ew" Command

In order to perform any actions on a network, the user must supply a network name to the system. This name is the same as the filename in which the network data is to be stored, but the user does not supply the extension to the filename. In effect, the network will create three files files with the same root name, but different extensions. These three files will be

1. "NetworkName.top", which contains the topology and network definition data associated with data entry items a. - g. discussed in Section 4.1.

2. "NetworkName.thp", which contains the relative throughput data for all traffic types defined by the user,

3. "NetworkName.pth", which contains the path sets associated with data entry items h. and i. discussed in Section 4.1.

These three files will be stored in whatever the current DOS directory was at the time of program initiation. When there is a need for the program to open these files, the program looks only in the current directory , i.e., the directory that was active at program initiation, or any other directories made available by the DOS "PATH" command.

In general, the program will prompt the user for a network file name if one has not been defined and the current requested action requires one. Once that name has been supplied to the program, it remains the current network file for all further actions unless the "N(ew" command is invoked. The "N(ew" command is the means by which the user can change from one network file to an unrelated one, if that is desired. Note that invoking the new command does not actually store or retrieve any data to memory, but only establishes that all further actions will refer to a different network file. The means by which data storage is accomplished in the program prevents any loss of data in any event: all relevant data for a network analysis is always stored as soon as it is created, so that errors on the part of the user concerning possible loss of data are minimized.

4.2.2.2 The "C(reate" Command

The "C(reate" command is used when a new network, not previously defined and stored to disk, is to be created. If no network name has yet been defined via the "N(ew" command, the user will be prompted to supply a network name. If a previous file of the same name already exists in the active directory on disk, then the user will be warned, and given the option to discontinue. (Continuing at this point will erase the previous file of the same name already on disk.) Once the filename has been selected, the "C(reate" function steps the user through all data entry associated withthe full definition of a network structure, with data entry being required for items a. - g. in Section 4.1, and the order of entry being in the order indicated.

The data needed to define a large network can be quite voluminous, so MMDESIGN provides certain shortcuts to the user to eliminate the entry of redundant or assumed values. This applies specifically to the following types of data.

1. Entry of error rates for all media and a specific traffic type can be eliminated if all such entries are identical with those for a previously defined traffic type. MMDESIGN asks if the current entries are like those for a previous traffic type, and if so, allows the user to input the traffic type index only. Then the previous error rate data is copied to the current traffic type.

2. Local arrival rates for the traffic type, i.e., the specific probabilities that a newly originated message will be associated with a given node, can also be copied from one traffic type to a later traffic type, by the mechanism described above.

3. The multiplex vectors, by which the composite channel for a given traffic type is defined (data item f. of Section 4.1) can be copied from one traffic type to another by the mechanism described above.

4. The topology of the network also is unique to each traffic type (i.e., each traffic type may adhere to a separate matrix of routing transition probabilities), but the routing matrix of a previous traffic type can be copied to the current type by the mechanism described above.

A final data entry economy is associated with the entry of specific routing probabilities for the routing transition probability matrix associated with a traffic type: namely, all entries of the probability matrix are initialized to zero, so the user need only enter the data associated with actual links in the network. For those data entries required, data is entered on a single line, as the origin node, destination node, and probability, in that order, separated by spaces and terminated by a carriage return.

The "C(reate" command steps the user through the input of all required data, looping through the traffic type specific data until all traffic types have been defined. When the data entry process is complete, it automatically verifies the consistency of the data, and provides a screen warning if any inconsistencies are found. This screen warning does not pinpoint the nature of the data inconsistency, however: the user should invoke the "MAIN/Edit/Verify" command in order to get a detailed account of where the inconsistencies were found.

A final important point is that the user need take no action to insure that a created network definition is stored to disk. The storage process is carried out simultaneously with data entry, and is automatically completed and the file closed at the termination of network creation.

4.2.2.3 The "E(dit" Command

Invoking "E(dit" at the MAIN menu level confronts the user with a new menu,

"E(dit, V(erify Q(uit:: "

The "MAIN/Edit/E(dit" command is used to modify a previous network definition stored in a network file. The file to be edited will be the current one, as shown in the Navigation/Status window, or, if none has been identified, the

"MAIN/Edit/E(dit" command will prompt for a file name. All of the network parameters in a file may be modified, with the exception of the number of network nodes, the number of traffic types, and the number of media types. (A later version of MMDESIGN will permit modification of these parameters also.)

Invoking the "MAIN/Edit/Edit" menu results in yet another menu of the form

" Edit functions are as follows:
0: Exit the edit function,
1: modify media bandwidths,
2: modify traffic type global arrival rates,
3. modify traffic type length,
4. modify traffic type acknowledgement length,
5: modify traffic type/medium type error rates,
6. modify traffic type station arrival rates,
7. modify traffic type medium multiplex vector,
8. modify traffic type station connectivity.

When the user invokes any of these choices except "0", the program prompts for information relating to the specific data type to be modified. Some of these choices, on the assumption that the user will wish to modify a multiplicity of them at one time, result in a menu of their own, which allows sequential modification of the data type in question, or a "Q(uit" option to return to the "MAIN/Edit/Edit" menu.

A final point concerning editing is that the user does not in fact edit the original data file during the actual edit session. Instead, a temporary file of the same root name, but with extension ".tmp" is created, and all editing changes are made to that file. When the user invokes the "MAIN/Edit/Edit/0" command to exit an editing session , the program provides the option of storing the edited data under the original file name, under a new file name, or abandoning the changes with no permanent file being created. If a new file name is chosen, it does not automatically become the active network of the program. It will be necessary for the user to use the "MAIN/New" command to select the new file for analysis.

Invoking the "MAIN/Edit/Verify" command provides the user with the opportunity to check the current network data file (i.e., the one displayed by the program in the Navigation/Status window) for consistency. The verify command provides specific outputs to screen and printer, if requested, indicating the nature and locations of the inconsistencies found. Data which contains inconsistencies will not provide reliable network performance metrics, and in fact may cause system crashes when computation based on it is attempted. The user must edit inconsistent data, and reverify it to insure that the results of analytical endeavors with MMDESIGN are meaningful

4.2.2.4 The "H(ardcopy" Command

The "H(ardcopy" command on the MAIN menu enables the user to get hardcopy output of the network definition data. Invoking the "H(ardcopy" command presents the user with another menu,

"Display G(lobal data, T(raffic type data, A(ll data, Q(uit: ".

The user can print out only that network data which is global (data items a. - d. in Section 4.1), only data that is specific to one traffic type (items e. - g. in Section 4.1), or the entire contents of the active network definition file. The printout is formatted so that the various data types contained in the file are easily distinguishable. If no data file is currently active, the program will prompt for one.

4.2.2.5 The "R(ecall" Command

The "R(ecall" command permits the user to bring into computer memory the global data from a network definition file, and the traffic type data in that file associated with one specific traffic type index. (Because the product-form analysis of even modest-sized networks requires a substantial body of data, the data for only one traffic type at a time is ever in memory. All analysis needed for that traffic type can be done from that data.) Invoking the "R(ecall" command establishes the recalled network and traffic type as the currently active data set in the program. The "R(ecall" command simultaneously brings any throughput data already computed for the traffic type into memory, (see 4.2.2.6 and Equation 2 - 3), so that the user may pursue the analysis of the traffic type.

4.2.2.6 The "T(hruput" Command

The "T(hruput" command (mispelled to save space in the menu line), is used to compute the relative throughputs for the network and traffic type currently active in the program. The relative throughputs for any traffic type are computed from Equation 2 - 3, and, once obtained, are the basis for all performance metrics available for analysis. Invoking the "T(hruputs" command begets the user another menu,

"C(ompute, D(isplay, P(rint, Q(uit:"

These menu entries permit the obvious actions to be performed, where the "P(rint" command may be used to print throughputs for one, or all, traffic types.

Since the throughput computation is the most intensive computation required for open network analysis, the results of a successful throughput computation are automatically stored to a file the name of which has the currently active network name as root, and the extension ".thp". Thus if a user wishes to terminate an analysis session, to be resumed at a later time, it will not be necessary to recompute these quantities, which computation may prove to be time-consuming for large networks.

4.2.2.7 The "P(aths" Command

The paths selected for analysis in the network can be chosen independently of the original network creation, editing, and throughput calculation processes. In the normal order of events, the analyst would define a network via the creation process, edit it if necessary to delete inconsistencies, and then compute the

throughputs associated with that network definition for the traffic types of interest. After those activities had been completed, the analyst could directly examine the metrics associated with the individual nodes of the network (see Section 4.2.2.8 below for a description of the available metrics). However, in order to understand the effect of the network structure on the end-to-end delay of traffic, the analyst must examine the delay times of multiple-node paths. If a particular end-to-end scenario of interest involves only one path, then the end-to-end delay is simply the sum of the nodal response time (i.e., queueing plus service delay) for the path.

However, if the end-to-end scenario involves an origin and destination connected by multiple paths, then the analyst might desire the mean delay for all traffic of a given type between the origin and destination, traveling by whatever path is available. This was discussed in detail in Section 3.5, where it was shown that the computation of expected delay requires in such case a weighted sum of path delays, for all paths regarded as routes between the origin and destination. The "P(aths" command permits the analyst to define sets of paths from a specific origin to a specific destination, such that any such set of paths will be taken as a collection over which expected end-to-end delay is to be computed. These sets can be created, edited, and verified using the "P(aths" command.

When the analyst invokes the "P(aths" command,the menu line

"C(reate, A(dd, D(elete, V(erify, H(ardcopy Q(uit: "

appears. The explanation of these options will be taken up in their order of appearance.

First, the "MAIN/Paths/C(reate command permits the analyst to establish a new set of paths between any origin and destination node, and to list the traffic types of interest for that set of paths. The user is informed (based on how many path sets have already been defined) of what integer index will be associated with this path set, and then is prompted for the number of paths to be included in the set. Following that, the user enters the individual paths, each as a sequence of nodes separated by spaces, all nodes of a given path being entered sequentially from origin to destination, separated by spaces, and terminated with the "ENTER" key.

The user is then prompted for the traffic types of interest relative to this path (i.e., the traffic types for which the expected aggregate end-to-end traffic delay is to be computed), which are to be entered separated by spaces, and terminated with the "ENTER" key. At the end of the path creation process, the program automatically chacks that the paths do indeed exist (i.e., all links of each path have an associated nonzero routing transition probability), and informs the analyst of any inconsistencies discovered.

The "MAIN/Paths/A(dd" and "MAIN/Paths/Delete" commands constitute the editing process for the library of stored path sets. The "A(dd" command allows the analyst to insert additional paths in existing path sets. The program prompts for the path set to which a new path is to be added, and, following the response, the analyst enters a path description in the same format as was described for the path set creation process. The additional path is automatically verified as valid

(in the same manner as for path creation), and the user is warned of inconsistency.

For the delete command, the analyst is prompted for a path set from which to delete a path, and then is shown a screen listing of the paths in the set. The user enters a path index for the path, as adduced from the screen listing of the paths.

Note that the path sets defined for a network are stored in a separate file, the root name of which is the network name, and the extension for which is ".pth". All creation and modiffication activities involving the path sets automatically update this file without user intervention.

The "MAIN/Paths/Verify " command can be called at any time by the analyst, and will either verify a specific path set as containing consistent paths, or will verify all existing path sets.

The "MAIN/Paths/Hardcopy" command permits the user to obtain a printer output of the contents of either a specific path set, or all path sets.

The "MAIN/Paths/Quit" command returns the user to the MAIN menu line.

4.2.2.8 The "M(etrics" Command

The "M(etrics" command allows the analyst to examine the various network peformance metrics which can be computed for the network. All of these metrics require first that the relative throughputs associated with Equation 2 - 3 and section 4.4.2.6 have been computed. All metrics related only to nodal performance can then be obtained directly: those relating to mean delay for traffic traveling along paths, or collections of paths, cannot be computed until the desired sets of paths have been defined via the "P(aths" command discussed in Section 4.4.2.7.

Invoking the "M(etrics" command presents the menu

"N(odes, P(aths, Q(ueue Length Density, H(ardcopy, Q(uit: ".

The "H(ardcopy" option does not prompt the user, but simply turns the printer on so that a hardcopy record of all metrics requested is provided. This hardcopy option remains activated until the "MAIN/Metrics/Quit" command is invoked. Hardcopy requested is formatted so that it is clear from the printout exactly what performance metrics have been supplied.

The "N(odes" command presents the user with a menu line

"S(ingle Node, A(ll nodes, Q(uit: ".

The user can exercise the "S(ingle Node" option to request the available performance metrics for a specific node, and may request the "A(ll Nodes" option to get a listing of the nodal performance metrics for all nodes. The screen listing of metrics scrolls, as does all ordinary screen output, so it is advisable to have invoked the "MAIN/Metrics/Hardcopy" option prior to invoking the "All Nodes"

option. In either the "All Nodes" or "Single Node" case, the metrics supplied for each node are

1. the throughput of the node for each traffic type,
2. the total throughput of the node for all traffic types combined,
3. the utilization of each composite channel at the node,
4. the utilization ot the individual media at the node,
5. the nodal response time for each traffic type at the node.

These measures will be given more precise mathematical definitions in Appendix B. Taken together, they provide a good diagnostic tool by which the analyst can examine bottlenecks in the network, and determine their causes.

Invoking the "MAIN/Metrics/Paths" command presents the user with another menu,

"S(ingle Path Set, A(ll Path Sets, Q(uit: "

The "S(ingle Path Set" option prompts the user to enter the identity of a single path set (path sets are discussed in connection with Section 4.2.2.7), and then the overall expected path delay for the aggregate of all paths in the set is computed and output to the screen and/or printer.

The "A(ll Path Sets" option outputs the same metrics as the "S(ingle Path Set" option, but does so for every path set which is in the paths file for the network. The delays are provided for each traffic type which was associated with the path set of interest at the time the path set was defined.

The numbers supplied in this case are just the mean delay time for transit of the traffic type(s) from the origin to the destination node. In a later version of the program, the computation of variance for that delay will also be supplied.

The "Q(ueue Length Density" command provides a more resolved look at the potential queueing bottlenecks in the system. When this command is invoked, the analyst is prompted for a node number and traffic type, and the program then computes and outputs the probability density of the queue length for that traffic type at that node. In theory, this density has infinitely many non-zero terms, but in practice, the terms are truncated when the queue length probabilities become less than 10^{-6}.

Invoking the "MAIN/Metrics/Quit" command returns the analyst to the MAIN program menu.

4.2.2.9 The "Q(uit" Command

Invoking the "Q(uit" command at the MAIN menu level exits the main program. Since all creation and editing processes in MMDESIGN are stored to files as they occur, the user may exit the program without first being concerned about data changes which may have been made during the program.

4.3 Summary and Further Directions

This completes the discussion of the program menus, and should provide the analyst with enough background to successfully exploit the power of MMDESIGN to examine the overall traffic flow in a network, and to seek better allocation of assets. The MMDESIGN program must be developed and used in prototype fashion over some range of test cases in order to fully understand its potential as an adjunct to network design by simulation. The second year of this study will be focused on studying such test cases with MMDESIGN, and, using the insight thus gained, creating automated capabilities within MMDESIGN to seek allocation of assets so as to achieve optimum traffic timeliness within the bandwidth constraints of the system.

5.0 REFERENCES

[1]. Aigner, Martin, *Combinatorial Search*, John Wiley & Scns, New York, NY, 1988

[2]. Nemhauser, George L., and Wolsey, Laurence A., *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, NY, 1988

[3]. Jackson, R. R. P., "Queueing Systems with Phase Type Service", J. R. Statistical Society, B18, 129-132

[4]. Baskett, F., Chandy, K., Muntz, R., Palacios, F., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", J. ACM, Vol. 22, No. 2, Apr '75, pp. 248 - 260

[5]. Lemoine, A. J., "Networks of Queues, a Survey of Equilibrium Analysis", Management Science, Vol. 24, No. 4, Dec. '77, pp. 464 - 481

[6]. Lavenberg, S.S. (Editor), *Computer Performance Modeling Handbook*, Academic Press, San Diego, CA, 1983

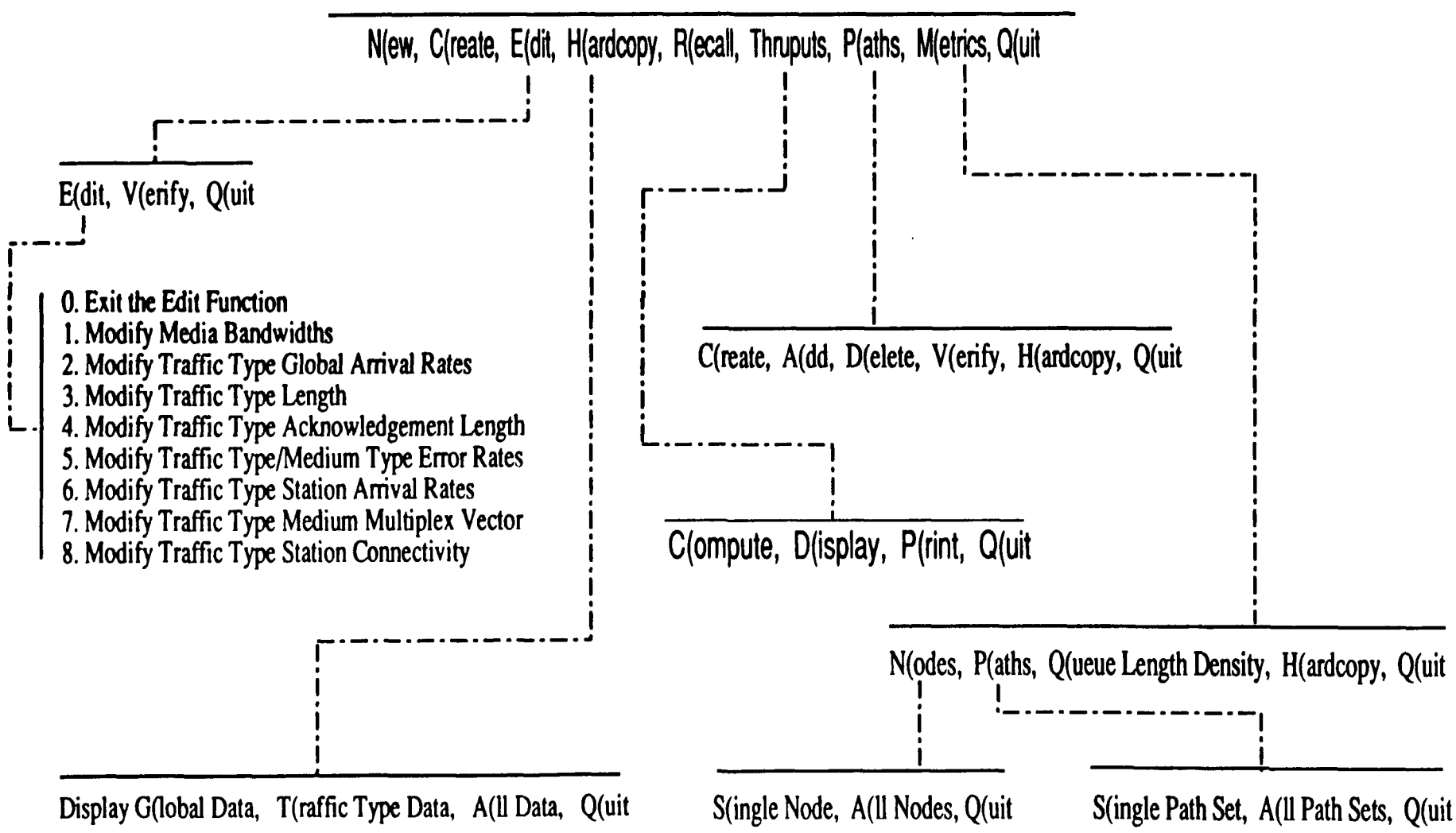
[7]. Akyildiz, I. F., "Exact Product-Form Solution for Queueing Networks with Blocking", IEEE Trans. on Computers, Vol. C-36, No. 1, Jan '87, pp 122-125

[8]. Conway, E. A., Georganas, N. D., "RECAL - A New, Efficient Algorithm for the Exact Analysis of Multiple Closed-Chain Queueing Networks", J. ACM, Vol. 33, No. 4, Oct. '86, pp. 768 - 791

[9]. DeSilva, E. D., Lavenberg, S. S., "Calculating Joint Queue -Length Distributions in Product-Form Queueing Networks", J. ACM, Vol. 36, No. 1, Jan '89, pp. 194 - 207

APPENDIX A

VIEW OF THE HIERARCHICAL MMDESIGN MENU STRUCTURE



APPENDIX B

THEORETICAL DESCRIPTION OF CLOSED-FORM MODELING FOR OPEN NETWORKS OF QUEUES

The general description of the context and limitations of closed-form network-of-queues models were discussed in Section 2. The techniques by which such modeling can be fruitfully applied to the design of multimedia communications networks were discussed in Section 3. In that section, an open network model was adopted for the study of the multimedia traffic type issues of communications. It is fortunate that the open network model is the germane model in this case, since the computational difficulties associated with that model are less severe than for closed networks. (Recall that an open network allows arrivals to and departures from the system, while a closed network does not: thus the customer count for a closed network does not change.)

This appendix will provide careful mathematical development of the essential expressions for the primary performance measures of open network models. The development presented in this appendix is the essential underlying mathematics on which the MMDESIGN program is based.

To begin this exposition, recall that the assumption underlying the success of the closed-form technique is that the network have a product-form. This assumption actually means that

1. the state of each node is expressible solely in terms of its current customer population, i.e., in terms of the numbers of customers of each type currently in queue and in service,

2. the state of the entire network is expressible exactly in terms of the individual states of the nodes.

The latter assumption is expressible in terms of a product of independent probabilities,

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\}$$

where

S_i is a vector representing the customer population at node i,

$Pr\{(S_1, S_2, \ldots, S_N)\}$ is the probability of the network having the aggregate state represented by the state vectors of the nodes, and

$Pr\{S_i\}$ is the probability that node i has the state represented by S_i.

These assumptions hold true provided that the routing in the network is probabilistic rather than deterministic, i.e., that each customer leaving a node has a probability of thereafter going to any other connected node. The routing probabilities are grouped in a routing transition probability matrix, which may take the form

P[(i,s), (j,t)] = probability of a customer of class s at node i transits to a customer of class t at node j .

However, the pairwise notation used above is inconvenient, so we opt instead to use a notation where each (node, customer class) pair takes on a single index notation, which we will call the customer type. In effect, each customer class has now been subdivided into many customer types. (There are then many more customer types than classes, but the matrix dimensions remain the same.)

Given this change of notation, the matric expression for routing transition probabilities becomes

P[c, d] = probability that a type c customer transits to a type d customer.

This matrix would be a square matrix, with dimension equal to the total number of customer types determined in this way. However, for open networks, we include the possibility that a message leaving processing at a node may be absorbed at that node. This effectively adds a "zero-th" customer type, which is included as a zero-th column of the matrix, P[c,0].

Together with this matrix, the arrival rates at the nodes determine the essential loading of the network, and from this, all measures of performance are derived. For reasons of mathematical convenience, the arrival process is expressed in terms of a global arrival rate per customer class, which is the total arrival rate for all customers of that class to all nodes, and a probability distribution which subdivides that arrival rate between all relevant nodes. The global arrival rate is taken to be Poisson (i.e., exponentially distributed interarrival times). Each customer class global arrival rate can in fact be dependent on the number of customers of that class already in the system: thus the arrival rate for customer class i could be expressed as a discrete function

$$\lambda_i(0),\ \lambda_i(1),\ \lambda_i(2),\ \ldots\ .$$

In the application of this theory to multimedia communications, there has been no need to consider variable arrival rates, so we will denote the arrival rate for customer class i simply by λ_i .

Now the global arrivals into customer class i are partitioned by a discrete probability density, say $p_i = (p_{i1}, p_{i2}, \ldots, p_{iN})$, where

p_{ij} = the probability that an arrived class i customer arrives at node j .

The actual consequences of this two-step description of arrival is actually equivalent to postulating Poisson arrivals for each customer class at each node, where the overall arrival rate for customer class i at node j becomes

$$\lambda_{ij} = p_{ij}\lambda_i. \qquad \text{(Equation B - 1)}$$

Converting over to customer types, where d is a customer type which is of customer class i, we will use the notation

P[0, d] = probability that a customer class global arrival of class i goes to customer type d.

In this notation, we are prepared to state what is the fundamental relationship from which all of the remaining performance measures flow, namely,

$$y_d = P[0,d] + \sum_{c \in C} y_c P[c,d] \qquad \text{(Equation B - 2)},$$

which expresses the relationship of the relative throughputs for the network. In this equation,

y_d = the relative throughput of customer type d, and
C = the set of all customer types (including the "0" type).

The equations represented by Equation B - 2 are a set of linear equations which, for open networks (i.e., at least one P[0, d] not equal to 0), are uniquely solvable for the quantities y_d, $d \in C$. In many instances, the equations in fact decompose into disjoint subsets of equations, because the potential classes and nodes that some subsets of customers might visit are restricted by routing limitations to less than the full sets of nodes and classes. Such subsets are called closed routing chains.

The quantities y_d are called throughputs because Equations B - 2 are effectively flow equations, and the y_d correspond to the total traffic intensity entering a node from all other sources. These throughputs are called "relative" because the equations do not involve in any way the global arrival rates to the system: however, the only effect of the global arrival rates is to scale the absolute traffic throughput values to some factor times the relative throughputs. There may be several of these customer "type" throughputs associated with a node, of course.

Once Equations B - 2 have been solved for the y_d, several derived performance measures for the nodes are available, as described below. For node j and customer type c, let

C_j = the set of all customer types passing through node j, and

$E[S_c]$ = the expected service demand of a customer of type c.

Then

$$y_{(j)} = \sum_{c \in C_j} y_c \qquad \text{(Equation B - 3)},$$

represents the total relative throughput of node j.

$$E[S_{(j)}] = \left[\sum_{c \in C_j} y_c E[S_c] \right] / y_{(j)} \qquad \text{(Equation B - 4)},$$

represents the expected service demand per customer on node j independent of customer type.

$$b_c = y_c E[S_c] \qquad \text{(Equation B - 5)},$$

represents the total customer type c service demand on node j.

$$b(j) = \sum_{c \varepsilon C_j} b_c \qquad \text{(Equation B - 6),}$$

represents the total expected service demand for node j.

Again, all of these numbers are relative quantities: they provide comparisons between nodes, but until they are multiplied by the absolute arrival rates, they do not provide absolute values for the indicated quantities. If λ_i represents the absolute arrival rate for type c customers (which are of class i), then we can express absolute arrival rate for type c customers as

$$\Lambda_c = \lambda_i y_c \qquad \text{(Equation B - 7),}$$

and we can express the type c absolute service demand as

$$\rho_c = \lambda_i b_c, \qquad \text{(Equation B - 8)}$$

and the absolute service demand on node j is

$$\rho'_{(j)} = \lambda_i b_{(j)} \qquad \text{(Equation B - 9).}$$

The above expressions do not reflect the fact that the service rates at the nodes can also be taken to be dependent on the number of customers already in queue at those nodes. However, we have no need of queue-dependent service rates in the MMDESIGN program, so we shall not consider the extra mathematics associated with that case.

We are now in a position to express the probability density for the queue length at a node. If μ_j is the service rate at node j, and n_j is the number of customers currently at node j, then

$$\text{Prob}\{n_j = q\} = [\, \rho'_{(j)} / \mu_j \,]^q . \text{Prob}\{n_j = 0\} \qquad \text{(Equation B - 10).}$$

For the FIFO queue, which is the only queue of interest in the MMDESIGN program, the latter factor is given by

$$\text{Prob}\{ n_j = 0 \} = 1 - (\lambda_i b_{(j)} / \mu_j) \qquad \text{(Equation B - 11).}$$

The latter term in the last equation is usually called the traffic intensity at a node, and we will denote it as

$$\rho_j = \lambda_i b_{(j)} / \mu_j \qquad \text{(Equation B - 12).}$$

From the above results for FIFO queues, it is possible to express the expected queue length in closed form, i.e.,

$$E[n_j] = \rho_j / (1 - \rho_j) \qquad \text{(Equation B - 13).}$$

The above results essentially provide the mathematical elements underlying the derivation of performance metrics for the MMDESIGN program.

APPENDIX C

MMDESIGN SOURCE CODE

The MMDesign source code is written in Borland's Turbo Pascal. The source code consists of a main program and two supporting units, as follows.

MMDesign.PAS -- the source code for the main program,

Data_IO.PAS -- the source code which supplies all the file handling and data manipulation activities for the main program

NetComp.PAS -- the source code which supplies the computational fucntions needed by the main program.

Note that MMDesign is an evolving program, and thus the source code supplied in this appendix will undoubtably be considerably expanded and altered following publication of this document.

```
PROGRAM MMDesign;              (*John R. Doner  8 August 1989*)

(*This program is the main implementation of the networks of queues theory
  as applied to the Multimedia Network Design Study.  Note that this code
  is in an evolutionary state, and as such includes partially implemented
  and unimplemented features.*)

USES Data_IO, NetComp, CRT;

(*************************************************************************
                DICTIONARY OF SIGNIFICANT PROGRAM VARIABLES

  BadFile       -- controls exit from a procedure if data file not available
  command       -- user input to any menu prompt
  IOWindow      -- denotes the main window on the screen for user input/output
  message       -- used to pass string to CenterText procedure
  NetDefined    -- specifies whether a network is currently in memory
  NetworkName   -- name of currently active network
  NoGo          -- general purpose flag, used variously in program
  Print         -- controls hardcopy output from the Data_IO.Verify procedure
  quit          -- controls exit from the main menu program loop
  TrafficIndex  -- denotes traffic type currently of interest

*************************************************************************)

VAR NetworkName, message                                  : STRING;
    TrafficIndex, i                                       : INTEGER;
    quit, BadFile, Print, NoGo, NetDefined                : BOOLEAN;
    command                                               : CHAR;
    IOWindow                                              : TEXT;

(*The following procedure provides the top-of-screen display on the screen,
  indicating the current status of the program.*)

PROCEDURE NewScreen(title: STRING);
VAR spaces: INTEGER;
    Xtop, YTop, XBottom, YBottom, BackColor, ForeColor, StatusColor: BYTE;
    Stat: TEXT;

BEGIN

  (*Open the status window and write display to it *)

  XTop := BYTE(1);
  YTop := BYTE(1);
  XBottom := BYTE(80);
  YBottom := BYTE(7);
  BackColor := BYTE(13);
  ForeColor := BYTE(14);
  StatusColor := BYTE(15);
  TextBackground(BackColor);
  TextColor(ForeColor);
  Window(Xtop, YTop, XBottom, YBottom);
  AssignCRT(Stat);
  REWRITE(Stat);
  ClrScr;
  Write(Stat,'A +++++++++++++++  AIRMICS MULTIMEDIA NETWORK DESIGN PROGRAM  ');
  WriteLn(Stat,'+++++++++++++++ A');
```

```
  WriteLn(Stat, 'I',' ':77, 'I');
  spaces := (40 - Length(NetworkName)) DIV 2;
  Write(Stat, 'R',' ':spaces, 'Current Data File: ');
  TextColor(StatusColor);
  Write(Stat, NetworkName);
  TextColor(ForeColor);
  Write(Stat, '  Traffic type:');
  TextColor(StatusColor);
  Write(Stat,TrafficIndex:3);
  TextColor(ForeColor);
  IF (2*spaces + 39 + Length(NetworkName)) < 79 THEN spaces := spaces + 1;
  WriteLn(Stat, ' ':spaces, 'R');
  WriteLn(Stat, 'M', ' ':77, 'M');
  spaces := (63 - Length(title)) DIV 2;
  Write(Stat, 'I',' ':spaces,'*** Menu: ');
  TextColor(StatusColor);
  Write(Stat, title);
  TextColor(ForeColor);
  IF (2*spaces + 16 + Length(title)) < 79 THEN spaces := spaces + 1;
  WriteLn(Stat,' ***', ' ':spaces, 'I');
  WriteLn(Stat, 'C', ' ':77, 'C');
  Write(Stat, 'S ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++');
  Write(Stat, '++++++++++++++++ S');
  CLOSE(Stat);
END(*NewScreen*);


(*The following procedure opens the main I/O window for data entry.*)

PROCEDURE MainWindow;
VAR  XTop, YTop, XBottom, YBottom, BackColor, ForeColor: BYTE;
BEGIN
  XTop := BYTE(1);
  YTop := BYTE(8);
  XBottom := BYTE(80);
  YBottom := BYTE(25);
  BackColor := BYTE(7);
  ForeColor := BYTE(1);
  TextBackGround(BackColor);
  TextColor(ForeColor);
  Window(XTop,YTop, XBottom, YBottom);
  AssignCRT(IOWindow);
  REWRITE(IOWindow);
  ClrScr;
  WriteLn(IOWindow)
END(*MainWindow*);

BEGIN(*MAIN PROGRAM*)

  (*Initialization of program status parameters*)

  NetworkName := 'Undefined';
  TrafficIndex := 0;
  NetDefined := FALSE;
  quit := FALSE;

  REPEAT
    NewScreen('MAIN');
    MainWindow;
```

```
Write(IOWindow, ' N(ew, C(reate, E(dit, H(ardcopy, R(ecall, T(hruputs,');
Write(IOWindow,' P(aths, M(etrics, Q(uit: ');
RESET(IOWindow);
ReadLn(IOWindow, command);
CASE command OF
 'N','n':
 BEGIN
   NewScreen('MAIN/New');
   MainWindow;
   Write(' Enter new network name: ');
   ReadLn(NetworkName);
   NetDefined := TRUE
 END(*CASE 'N'*);
 'C','c':
  BEGIN
    NoGo := FALSE;
    NewScreen('MAIN/Create Network');
    MainWindow;
    Write(' Enter the filename in which network data is to be stored: ');
    ReadLn(NetworkName);
    NewScreen('MAIN/Create Network');
    MainWindow;
    NetDefined := TRUE;
    CreateNetwork(NetworkName);
    TrafficIndex := NumberTrafficTypes
  END(*CASE Create*);
  'E','e':
  BEGIN
    REPEAT
      NewScreen('MAIN/Edit');
      MainWindow;
      IF NOT NetDefined THEN
      BEGIN
        Write(' Enter name of file containing network data: ');
        ReadLn(NetworkName);
        NetDefined := TRUE;
        NewScreen('MAIN/Edit');
        MainWindow
      END;
      Write(' E(dit,  V(erify,  Q(uit: ');
      ReadLn(command);
      CASE command OF
        'E','e':
        BEGIN
          NewScreen('MAIN/Edit Network Data');
          MainWindow;
          EditNetwork(NetworkName);
        END(*CASE 'E'*);
        'V','v':
        BEGIN
          NewScreen('MAIN/Edit/Verify Network Data');
          MainWindow;
          WriteLn;
          WriteLn('                    **** Network Data Verification ****');
          WriteLn;
          Write(' Is hardcopy output desired? (y/n): ');
          ReadLn(command);
          WriteLn;
          IF command = 'y' THEN Print := TRUE ELSE Print := FALSE;
```

```
        IF NOT Verify(NetworkName, Print) THEN
        BEGIN
          WriteLn;
          BEEP;
          TextAttr := BlinkOn OR TextAttr;
          WriteLn('              **** WARNING: data must be edited before use. ****');
          TextAttr := BlinkOff AND TextAttr;
          WriteLn
        END
        ELSE Write(' Network data passes all consistency tests.');
        Write(' Press any key to continue.');
        ReadLn
      END(*CASE 'V'*);
      'Q', 'q': command := 'q'
    END(*MAIN/Edit CASES*)
  UNTIL command = 'q'
END(*CASE Edit*);
'H','h':
BEGIN
  NewScreen('MAIN/Hardcopy');
  MainWindow;
  IF NOT NetDefined THEN
  BEGIN
    Write(' Enter name of network data file: ');
    ReadLn(NetworkName);
    NetDefined := TRUE
  END;
  WHILE command <> 'q' DO
  BEGIN
    NewScreen('MAIN/Hardcopy');
    MainWindow;
    BadFile := FALSE;
    WriteLn;
    Write(' Display G(lobal data, T(raffic type data, A(ll data, Q(uit: ');
    ReadLn(command);
    CASE command OF
      'G','g':
      BEGIN
        IF NOT DisplayNetwork(0, NetworkName) THEN BadFile := TRUE
      END(*CASE 'g'*);
      'T','t':
      BEGIN
        WriteLn;
        Write(' Enter traffic type for which hardcopy is desired: ');
        ReadLn(i);
        IF NOT DisplayNetWork(i, NetworkName) THEN BadFile := TRUE
      END(*CASE 't'*);
      'A','a':
      BEGIN
        IF NOT DisplayNetwork(0, NetworkName) THEN BadFile := TRUE;
        FOR i := 1 TO NumberTrafficTypes DO
          IF NOT DisplayNetwork(i,NetworkName) THEN BadFile := TRUE
      END(*CASE 'a'*);
      'q','Q':
      BEGIN

      (*Make sure that original data is back in memory*)

       IF TrafficIndex <> 0 THEN
```

```
        IF RetrieveNetwork(TrafficIndex, NetworkName) THEN ;
       command := 'q'
      END(*CASE 'q'*)
    END(*CASE command/Display menu*);
    IF BadFile THEN
    BEGIN
      BEEP;
      TextAttr := TextAttr OR BlinkOn;
      WriteLn;
      Write(' The specified data file cannot be opened:');
      Write(' press any key to continue.');
      ReadLn;
      TextAttr := TextAttr AND BlinkOff;
      ClrScr
    END(*IF BadFile*)
  END(*WHILE command ...*)
END(*CASE Display*);
'R','r':
BEGIN
  NewScreen('MAIN/Retrieve');
  MainWindow;
  WriteLn;
  Write(' Enter disk file name for network: ');
  ReadLn(NetworkName);
  Write(' Enter traffic type of interest: ');
  ReadLn(TrafficIndex);
  WriteLn;
  IF NOT RetrieveNetwork(TrafficIndex, NetworkName) THEN
    Write(' Retrieval from disk failed: ')
  ELSE
  BEGIN
    Write(' Network data loaded to memory: ');
    NetDefined :=TRUE
  END(*IF NOT RetrieveNetwork... ELSE...*);
  Write('press any key to exit to MAIN Menu.');
  ReadLn
END(*CASE Retrieve*);
'T','t':
BEGIN
  REPEAT
    NewScreen('MAIN/Thruputs');
    MainWindow;
    Write(' C(ompute, D(isplay, P(rint, Q(uit: ');
    ReadLn(command);
    CASE command OF
      'C','c':
      BEGIN
        NewScreen('MAIN/Thruputs/Compute');
        MainWindow;
        Write('Compute A(ll or a S(pecific throughput? ');
        ReadLn(command);
        CASE command OF
          'A','a':
          BEGIN
            FOR i := 1 TO NumberStations DO
              IF NOT SolveThruputs(i, NetworkName) THEN
              BEGIN
                BEEP;
                STR(i:3,message);
```

```
            message := 'Thruput computation failed for traffic type'
                    + message;
            CenterText(message)
          END
          ELSE Write('Thruput computed for traffic type ',i:3);
        Write(': press any key')
      END(*CASE 'A'*);
      'S','s':
      BEGIN
        Write('Enter traffic type for which to compute throughputs: ');
        ReadLn(i);
        IF NOT SolveThruputs(i, NetworkName) THEN
        BEGIN
          BEEP;
          Write('Solution for throughputs failed:')
          END
        ELSE Write('Throughputs computed:');
        Write(' press any key to continue. ');
        ReadLn
      END(*CASE 'S'*)
    END(*CASE command...*)
  END(*CASE 'C'*);
  'D','d':
  BEGIN
    WriteLn(' Function not implemented: press any key to continue.');
    ReadLn
  END;
  'P','p':
  BEGIN
    WriteLn(' Function not implemented: press any key to continue.');
    ReadLn
  END;
  'Q','q': command := 'q';
 END(*MAIN/Thruput CASES*);
UNTIL command = 'q';
END(*CASE 'T'*);
'P','p':
BEGIN
 REPEAT
  NewScreen('MAIN/Paths');
  MainWindow;
  Write(' C(reate, A(dd, D(elete, V(erify, H(ardcopy, Q(uit: ');
  ReadLn(command);
  CASE command OF
    'C','c':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
    'A','a':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
    'D','d':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
```

```
          'V','v':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'H','h':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'Q','q': command := 'q'
        END(*CASE command...*)
      UNTIL command = 'q'
    END(*CASE Paths*);
    'M', 'm':
    BEGIN
      REPEAT
        NewScreen('MAIN/Metrics');
        MainWindow;
        Write(' N(odes,  P(aths, Q(ueue length density,  H(ardcopy,  E(nd: ');
        ReadLn(command);
        CASE command OF
          'N','n':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'P','p':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'Q','q':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'H','h':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'E','e': command := 'e'
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE Metrics*);
    'Q','q':  quit := TRUE
  END(*MAIN Menu CASES*)
UNTIL quit
END(*Main Program*).
```

```
UNIT Data_IO;                (*John R. Doner  20 July 1989*)

(*************************************************************************
  This unit supplies all of the procedures, data types and variables
  needed to manage the input data associated with a full network
  description as required by the AIRMICS MultiMedia Network Design
  Closed-Form Queueing Model.
*************************************************************************)

INTERFACE

  USES DOS, CRT;

  CONST AnyFile = $3F;          (*Used by DOS FindFirst() procedure.*)
        BlinkOn = BYTE(132);    (*Used to blink warning messages*)
        BlinkOff = BYTE(123);   (*Turn off blinking*)
        MaxNodes = 30;          (*Maximum number of nodes: memory limited.*)
        MaxMediumTypes = 3;     (*Maximum number of media types.*)
        MaxTrafficTypes = 3;    (*Maximum number of traffic types.*)
        FormFeed = CHR(12);     (*Formfeed control code for printer.*)

(*************************************************************************
          DICTIONARY OF SIGNIFICANT PROGRAM VARIABLES AND TYPES

  AckLength             -- length of the acknowledgement for current traffic
  ErrorRates[i]         -- missed message rates (MMR) of current traffic type
                           relative to each medium
  GlobalArrivalRate     -- network-wide arrival rate of current traffic type
  MediumBandWidth[i]    -- bandwidths (bits/second) of the available media
  NumberMediumTypes     -- total number of media available in the network
  NumberStations        -- total number communications nodes in network
  NumberTrafficTypes    -- total number of traffic types in system
  StationMultiplex[i,j] -- vectors used to media multiplex traffic at stations
  StationSourceRate[i]  -- relative traffic origination rate for station i
  StationThruPuts[i,j]  -- the relative station throughputs for each traffic
                           type (calculated from input data)
  StoredData            -- a type used by the Fetch() procedure to determine
                           which type of data is to be copied to the current
                           data input process from previously stored data
  Topology[i,j]         - traffic routing transition probability matrix
  TrafficLength         - length of the traffic type currently being considered

*************************************************************************)

  TYPE StoredData = (errors, arrivals, multiplex, connectivity);

  VAR NumberStations,
      NumberMediumTypes,
      NumberTrafficTypes     : INTEGER;
      Topology               : ARRAY[1..MaxNodes, 0..MaxNodes] OF REAL;
      StationMultiplex       : ARRAY[1..MaxNodes, 1..MaxMediumTypes] OF REAL;
      GlobalArrivalRate,
      TrafficLength,
      AckLength              : REAL;
      StationSourceRate      : ARRAY[1..MaxNodes] OF REAL;
      StationThruputs        : ARRAY[1..MaxTrafficTypes, 1..MaxNodes] OF REAL;
      MediumBandWidth        : ARRAY[1..MaxMediumTypes] OF REAL;
      ErrorRates             : ARRAY[1..MaxMediumTypes] OF REAL;
```

```
(*************************************************************************
  The following procedure formats and centers the string variable passed to it
  and writes it to the screen.
*************************************************************************)

  PROCEDURE CenterText(message: STRING);


(*************************************************************************
  The following procedure emits a short tone from the speaker to alert
  the user to warning messages on the screen.
*************************************************************************)

  PROCEDURE BEEP;


(*************************************************************************
  The following function adds ".top" to the input filename, and then
  retrieves the data from the so-named disk file containing a network
  description.  "TrafficIndex" designates for which traffic type the data is
  to be retrieved.  The input file should be closed when this procedure is
  entered, and will be closed at procedure exit.  TRUE is returned only if
  the retrieve operation is successful.
*************************************************************************)

  FUNCTION RetrieveNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
  The following function stores the computed relative throughput data to
  the file named by the input FileName, after adding the extension ".thp"
  to the filename.  TRUE is returned only if the store operation was
  successful. NOTE: the throughputs should be stored under the same filename
  as is the network data.  The two files will have the same base name, but
  extensions ".top" for the network data and ".thp" for the throughput data.
  The output file should be closed when the procedure is called, and will be
  closed at procedure exit.
*************************************************************************)

  FUNCTION StoreThruPuts(FileName: STRING) BOOLEAN;


(*************************************************************************
  The following function retrieves throughput data from a disk file,
  after adding the extension ".thp" to the input filename.  TRUE is returned
  only if the operation is successful. The file should be closed upon
  entry to the procedure, and will be closed at procedure exit.
*************************************************************************)

  FUNCTION RetrieveThruPuts(FileName: STRING) BOOLEAN;


(*************************************************************************
  The following procedure locates specific data fields within the "DataFile"
  file and retrieves them, writing them into the analogous program variables
  representing the data type retrieved.  Any previous value of that data type
  extant in memory is overwritten.  The data retrieved is of the type
  requested by the input "DataType", and the specific instantiation retrieved
  is that associated with the traffic type denoted by "TrafficIndex" or
```

```
  "TrafficIndex" and "station". For fetches of all data types except station
  multiplex data, the data returned is that associated with a previously
  defined traffic type: "station" is ignored during such a request. For
  station multiplex data, the data type returned is for the current traffic
  type and a previous station. TRUE is returned only if the data retrieval
  was successful. Fetch neither opens nor closes the data file, and leaves
  the file pointer at its original position upon exit.
**************************************************************************)

  PROCEDURE FetchData(TrafficIndex, station: INTEGER; DataType: StoredData;
                      VAR DataFile: FILE);


(**************************************************************************
  The following function checks for three required types of consistency
  in the current data. First, it checks that the rows of the Topology matrix
  for the current traffic type sum to one. Then it checks to see that the
  sum of the StationSourceRates is one, and finally it checks to see that
  no medium at any station is required to carry more than 100% of its bandwidth
  in traffic as a result of the media multiplexing scheme. Due to the
  inexactness of digital computation, the first two checks actually only
  require that the sums be within 1% of 1.0. When that is the case, the
  summed values are normalized to obtain the maximum precision available
  within the limits of the type REAL floating point number format. FALSE is
  returned if any constraint is not met, and an internal message is written
  to the screen indicating the nature of the inconsistency. The data file
  should be closed upon procedure entry, and will be closed at procedure exit.
  The input variable "HardCopy", when true cause printout of the verify data
  to the printer.
**************************************************************************)

  FUNCTION Verify(FileName: STRING; HardCopy: BOOLEAN): BOOLEAN;


(**************************************************************************
  The following procedure prompts the user for all input data required for
  the definition of a communications network. A disk file is used for output
  of the data. The file, if already in existence, should be closed before
  procedure entry, and will be closed at procedure exit.
**************************************************************************)

  PROCEDURE CreateNetwork(FileName: STRING);


(**************************************************************************
  The following procedure prompts the user for desired changes to the net-
  work. Use of this procedure is predicated on the existence of an already
  defined network in the current directory, under the input name "FileName".
  The EditNetwork procedure makes a copy of that network file on disk, and
  makes all alterations to the copy. At the end of the edit session, the
  user may choose whether the copy is to replace the original, or is to be
  stored under a separate name. The original file to be edited should be
  closed at procedure entry, and will be closed at procedure exit.
**************************************************************************)

  PROCEDURE EditNetwork(FileName: STRING);
```

```
(*************************************************************************
  The following procedure provides a hardcopy output of all the input data
  required to define a network and a single traffic type. Entry of a "0" as
  the traffic index results in display of the network wide variables. For
  all entrys of legitimate values of "TrafficIndex", the information specific
  to that traffic type will be printed. The file to be used should be closed
  upon procedure entry, and will be closed at procedure exit. TRUE is
  returned only if the file can be opened.
*************************************************************************)

  FUNCTION DisplayNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
  The following procedure displays the absolute messsage throughputs of the
  nodes for traffic type designated by "TrafficIndex". The filename should
  be the same as that under which the network topology data was stored.
  The file should be closed upon procedure entry, and will be closed at
  procedure exit. TRUE is returned only if the file is found and successfully
  opened.
*************************************************************************)

  FUNCTION DisplayThruputs(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
*************************************************************************)
IMPLEMENTATION

  PROCEDURE CenterText(message: STRING);
  VAR spaces: INTEGER;

  BEGIN
    spaces := (69 - Length(message)) DIV 2;
    message := '**** ' + message + ' ****';
    WriteLn(' ':spaces, message)
  END(*CenterText*);

  (*The following procedure gets the global data needed to size the file. This
    procedure is not available to calling programs.*)

  PROCEDURE GetGlobal(VAR DataFile: FILE);
  VAR i: INTEGER;
  BEGIN
    SEEK(DataFile, 0);
    Blockread(DataFile, Numberstations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes))
  END(*GetGlobal*);


  PROCEDURE BEEP;
  BEGIN
    SOUND(1000);
    DELAY(200);
    NOSOUND
  END;
```

```
FUNCTION RetrieveNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;
VAR i, j: INTEGER;
    StartPoint, PreLoop, LoopSize: LONGINT;
    DataFile: FILE;
    FileInfo: SearchRec;

BEGIN
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    RetrieveNetwork := FALSE;
    EXIT
  END
  ELSE
  BEGIN
    ASSIGN(DataFile, FileName);
    RESET(DataFile, 1);
    BlockRead(DataFile, NumberStations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
    IF TrafficIndex > NumberTrafficTypes THEN
    BEGIN
      Write('This traffic type does not exist in ', FileName,': ');
      Write('press any key to continue.');
      ReadLn;
      CLOSE(DataFile);
      RetrieveNetwork := FALSE;
      EXIT
    END(*IF TrafficIndex > ...*);
    PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
    LoopSize := (3 + NumberStations + (NumberStations + 1)*NumberMediumTypes
              + SQR(NumberStations))*SIZEOF(REAL);
    StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
    SEEK(DataFile, StartPoint);
    BlockRead(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
    BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
    BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
    BlockRead(DataFile, AckLength, SIZEOF(AckLength));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, ErrorRates[i], SIZEOF(ErrorRates[i]));
    FOR i := 1 TO NumberStations DO
      BlockRead(DataFile, StationSourceRate[i], SIZEOF(StationSourceRate[i]));
    FOR i := 1 TO NumberStations DO
      FOR j := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, StationMultiplex[i,j], SIZEOF(StationMultiplex[i,j]));
    FOR i := 1 TO NumberStations DO
      FOR j := 0 TO NumberStations DO
        BlockRead(DataFile, Topology[i,j], SIZEOF(Topology[i,j]));
    RetrieveNetwork := TRUE;
    CLOSE(DataFile)
  END(*IF DOSError...ELSE...*)
END(*RetrieveNetwork*);
```

```
  FUNCTION StoreThruputs(FileName: STRING): BOOLEAN;
  VAR i, j: INTEGER;
      ThruputFile: FILE;

  BEGIN
    FileName := FileName + '.thp';
    ASSIGN(ThruputFile, FileName);
{$I-}
    REWRITE(ThruputFile, 1);
{$I+}
    IF IOResult <> 0 THEN
    BEGIN
      Write('Throughput storage file could not be opened: press any key to');
      WriteLn(' continue.');
      ReadLn;
      StoreThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN

     (*Insert the following data to make the file self-contained.*)

      BlockWrite(ThruputFile, NumberStations, SIZEOF(NumberStations));
      BlockWrite(ThruputFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));

      FOR i := 1 TO NumberTrafficTypes DO
        FOR j := 1 TO NumberStations DO
          BlockWrite(ThruputFile, StationThruputs[i, j],
            SIZEOF(StationThruputs[i,j]));
      StoreThruPuts := TRUE;
      CLOSE(ThruputFile)
    END(*IF IOResult... ELSE...*)
  END(*StoreThruputs*);

  FUNCTION RetrieveThruputs(FileName: STRING): BOOLEAN;
  VAR i, j: INTEGER;
      ThruputFile: FILE;
      FileInfo: SearchRec;

  BEGIN
    FileName := FileName + '.thp';
    FindFirst(FileName, AnyFile, FileInfo);
    IF DOSError <> 0 THEN
    BEGIN
      Write('Throughput file ', Filename,' not found: press any key to ');
      WriteLn('continue.');
      ReadLn;
      RetrieveThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      ASSIGN(ThruputFile, FileName);
{$I-}
      REWRITE(ThruputFile, 1);
{$I+}
      IF IOResult <> 0 THEN
      BEGIN
```

```
      Write('Throughput storage file could not be opened: press any key to');
      WriteLn(' continue.');
      ReadLn;
      RetrieveThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      BlockRead(ThruputFile, NumberStations, SIZEOF(NumberStations));
      BlockRead(ThruputFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
      FOR i := 1 TO NumberTrafficTypes DO
        FOR j := 1 TO NumberStations DO
          BlockWrite(ThruputFile, StationThruputs[i, j],
                SIZEOF(StationThruputs[i,j]));
      RetrieveThruputs := TRUE;
      CLOSE(ThruputFile)
    END(*IF IOResult...ELSE...*)
  END(*IF DOSError ... ELSE...*)
END(*RetrieveThruputs*);

PROCEDURE FetchData(TrafficIndex, station: INTEGER; DataType:
                StoredData; VAR DataFile: FILE);

VAR i, j: INTEGER;
    FileStart, PreLoop, LoopSize, InLoop, StartPoint: LONGINT;

BEGIN
  FileStart := FilePos(DataFile);
  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations + SQR(NumberStations) +
              NumberStations*NumberMediumTypes)*SIZEOF(REAL);
  StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
  CASE DataType OF
    errors:
    BEGIN
      InLoop := StartPoint + 3*SIZEOF(REAL);
      Seek(DataFile, InLoop);
      FOR i := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, ErrorRates[i], SIZEOF(ErrorRates[i]))
    END(*errors*);
    arrivals:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes)*SIZEOF(REAL);
      Seek(DataFile, InLoop);
      FOR i := 1 TO NumberStations DO
        BlockRead(DataFile, StationSourceRate[i], SIZEOF(StationSourceRate[i]))
    END(*arrivals*);
    multiplex:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes + NumberStations +
              (station - 1)*NumberMediumTypes)*SIZEOF(REAL);
      SEEK(DataFile, InLoop);
      FOR j := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, StationMultiplex[station,j],
                SIZEOF(StationMultiplex[station,j]))
    END(*multiplex*);
    connectivity:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes
```

```
            + NumberStations*(NumberMediumTypes + 1))*SIZEOF(REAL);
      SEEK(DataFile, InLoop);
      FOR i := 1 TO NumberStations DO
        FOR j := 0 TO NumberStations DO
          BlockRead(DataFile, Topology[i,j], SIZEOF(REAL));
    END(*connectivity*)
  END(*CASES*);
  Seek(DataFile, FileStart)
END(*FetchData*);

FUNCTION Verify(FileName: STRING; HardCopy: BOOLEAN): BOOLEAN;
VAR i, j, k, m: INTEGER;
    sum: REAL;
    message, message2: STRING;
    Transitions, Capacity, SourceRates: BOOLEAN;
    DataFile: FILE;
    FileInfo: SearchRec;
    MultiplexSums: ARRAY[1..MaxNodes,1..MaxMediumTypes] OF REAL;
    lst: TEXT;

BEGIN
(*Open file.*)

  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN

  (*Can't open, so make a graceful exit.*)

    Write('File ', FileName, ' not found: press any key to continue.');
    ReadLn;
    EXIT
  END
  ELSE
  BEGIN
    IF HardCopy THEN
    BEGIN
      ASSIGN(lst, 'prn');
      REWRITE(lst)
    END;

    (*Open file, and get the essential parameters at the top of file.*)

    ASSIGN(DataFile, FileName);
    RESET(DataFile, 1);
    BlockRead(DataFile, NumberStations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));

    IF HardCopy THEN
    BEGIN
      WriteLn(lst, 'Verification of routing transition probabilities:');
      WriteLn(lst)
    END;
    Transitions := TRUE;
    FOR i := 1 TO NumberTrafficTypes DO
```

```
  BEGIN

(*First check for consistency of topology information.*)

    FetchData(i, 0, connectivity, DataFile);
    FOR j := 1 TO NumberStations DO
    BEGIN
      sum := 0.0;
      FOR k := 0 TO NumberStations DO sum := sum + Topology[j, k];
      IF ABS(sum - 1.0) <= 0.01 THEN
        FOR m := 0 TO NumberStations DO Topology[i,m] := Topology[i,m]/sum
      ELSE
      BEGIN
        Transitions := FALSE;
        Str(i:3, message);
        message := 'WARNING: routing probability data, traffic type '
          + message + ' is inconsistent.';
        CenterText(message);
        IF HardCopy  THEN
        BEGIN
          Write(lst, '  Routing probability data for traffic type ',i:3);
          WriteLn(lst, ', station ',j:3, ' sums to ', sum:5:3)
        END
      END;
    END(*FOR j ...*)
  END(*FOR i...*);
  WriteLn;
  IF (HardCopy AND Transitions) THEN
    WriteLn(lst, 'No inconsistencies were found.');

(*Next, check for consistency of source rate data.*)

  IF HardCopy THEN
  BEGIN
    WriteLn(lst);
    WriteLn(lst);
    WriteLn(lst, 'Verification of station arrival rate data:');
    WriteLn(lst)
  END;
  SourceRates := TRUE;
  FOR i := 1 TO NumberTrafficTypes DO
  BEGIN
    sum := 0.0;
    FetchData(i, 0, arrivals, DataFile);
    FOR j := 1 TO NumberStations DO  sum := sum + StationSourceRate[j];
    IF ABS(sum - 1.0) <= 0.01 THEN
      FOR j := 1 TO NumberStations DO
        StationSourceRate[j] := StationSourceRate[j]/sum
    ELSE
    BEGIN
      SourceRates := FALSE;
      Str(i:3, message);
      message := 'WARNING: station arrival rates for traffic type'
        +  message + ' are inconsistent ';
      CenterText(message);
      IF HardCopy THEN
      BEGIN
        Write(lst, '  Station arrival rates for traffic type ', i:3);
        WriteLn(lst, ' sum to ',sum:5:2)
```

```
      END
    END(*FOR j ...*)
  END(*FOR i...*);
  WriteIn;
  IF (HardCopy AND SourceRates) THEN
    WriteLn(lst, 'No inconsistencies were found.');

 (*Finally, sum up the traffic type demands on the available channels.*)

  IF HardCopy THEN
  BEGIN
    WriteLn(lst);
    WriteLn(lst);
    WriteLn(lst, 'Verification of channel capacity constraints: ');
    WriteLn(lst)
  END;
  Capacity := TRUE;
  FOR i := 1 TO NumberStations DO
    FOR j := 1 TO NumberMediumTypes DO MultiPlexSums[i,j] := 0.0;
  FOR i := 1 TO NumberTrafficTypes DO
    FOR j := 1 TO NumberStations DO
    BEGIN
      FetchData(i, j, multiplex, DataFile);
      FOR k := 1 TO NumberMediumTypes DO
        MultiplexSums[j,k] := MultiplexSums[j,k] + StationMultiplex[j, k]
    END(*FOR i, j, k...*);
    FOR j := 1 TO NumberStations DO
      FOR k := 1 TO NumberMediumTypes DO
        IF MultiplexSums[j, k] > 1.0 THEN
        BEGIN
          STR(k:3, message);
          STR(j:3, message2);
          message := 'WARNING: channel capacity for medium ' + message +
            ', station ' + message2 + ' exceeded.';
          CenterText(message);
          IF HardCopy THEN
          BEGIN
            Write(lst, '  Multiplexed channel capacity for medium ',k:3);
            WriteLn(lst,' at station ',j:3, ' sums to ',MultiplexSums[j,k]:5:3)
          END;
          Capacity := FALSE
        END(*FOR i, j, IF...*);
   IF (HardCopy AND Capacity) THEN
    WriteLn(lst, 'No inconsistencies were found.');
  IF HardCopy THEN CLOSE(lst);
  Verify := Transitions AND Capacity AND SourceRates
 END(*IF DOSError...ELSE...*)
END(*Verify*);

PROCEDURE CreateNetwork;
VAR i, j, k, index: INTEGER;
  sum: REAL;
  FullName, message: STRING;
  command: CHAR;
  DataFile: FILE;
  FileInfo: SearchRec;

BEGIN
```

```
(*File name entry*)
  REPEAT
    CenterText('NETWORK CREATION');
    command := 'y';
    WriteLn;
    WriteLn('Data will be stored to disk as it is entered.');
    WriteLn;
    FullName := FileName + '.top';
    FindFirst(FullName, AnyFile, FileInfo);
    IF DOSError = 0 THEN
    BEGIN
      WriteLn;
      BEEP;
      TextAttr := TextAttr OR BlinkOn;
      CenterText('WARNING: Like-named file already on disk will be destroyed.');
      TextAttr := TextAttr AND BlinkOff;
      WriteLn;
      Write('      Proceed anyway? (y/n): ');
      ReadLn(command);
      ClrScr
    END
    ELSE
      command := 'y';
  UNTIL command = 'y';
  ASSIGN(DataFile, FullName);
  REWRITE(DataFile, 1);

(*NumberStations*)
  REPEAT
    Write('Enter number of communications stations (not exceeding ');
    Write(MaxNodes:3,'): ');
    ReadLn(NumberStations)
  UNTIL NumberStations <= MaxNodes ;
  BlockWrite(DataFile, NumberStations, SIZEOF(INTEGER));
  WriteLn;

(*NumberMediaTypes*)
  REPEAT
    Write('Enter number of media types (not exceeding ');
    Write(MaxMediumTypes:3,'): ');
    ReadLn(NumberMediumTypes)
  UNTIL NumberMediumTypes <= MaxMediumTypes;
  BlockWrite(DataFile, NumberMediumTypes, SIZEOF(INTEGER));
  WriteLn;

(*MediumBandWidth[.]*)
  FOR i := 1 TO NumberMediumTypes DO
  BEGIN
    Write('  Enter bandwidth of medium type ', i:3, ' (Kbits/second): ');
    ReadLn(MediumBandWidth[i]);
    BlockWrite(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
  END;
  WriteLn;

(*NumberTrafficTypes*)
  REPEAT
    Write('Enter number of traffic types (not exceeding ');
    Write(MaxTrafficTypes:3, '): ');
    ReadLn(NumberTrafficTypes)
```

```
  UNTIL NumberTrafficTypes <= MaxTrafficTypes;
  BlockWrite(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
  WriteLn;

(*All remaining data is traffic type dependent, and so will be entered
  for each traffic type.*)

  FOR i := 1 TO NumberTrafficTypes DO
  BEGIN
    ClrScr;
    WriteLn;
    BEEP;
    TextAttr := TextAttr OR BlinkOn;
    Str(i:3, message);
    message := 'Data input for traffic type ' + message;
    CenterText(message);
    TextAttr := TextAttr AND BlinkOff;

(*GlobalArrivalRate*)
    WriteLn;
    Write('Enter the network-wide traffic type arrival rate  (messages/sec.): ');
    ReadLn(GlobalArrivalRate);
    BlockWrite(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
    WriteLn;

(*TrafficLength*)
    Write('Enter mean message length (in bits) for traffic type: ');
    ReadLn(TrafficLength);
    BlockWrite(DataFile, TrafficLength, SIZEOF(TrafficLength));
    WriteLn;

(*AckLength*)
    Write('Enter mean length (bits) for acknowledgement message (0 if none sent): ');
    ReadLn(AckLength);
    BlockWrite(DataFile, AckLength, SIZEOF(AckLength));
    WriteLn;

(*ErrorRates*)
    ClrScr;
    CenterText('Entry of traffic type MMR for each medium type');
    WriteLn;
    Write('Copy previous missed message rates? (n/y): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, errors, DataFile)
    END
    ELSE
      FOR j := 1 TO NumberMediumTypes DO
      BEGIN
        Write('Enter missed message rate for medium ',j:2,': ');
        ReadLn(ErrorRates[j])
      END;
    FOR j := 1 TO NumberMediumTypes DO
      BlockWrite(DataFile, ErrorRates[j], SIZEOF(ErrorRates[j]));
```

```
    WriteLn;

  (*StationSourceRate*)
    ClrScr;
    CenterText('Station relative arrival rates for traffic type');
    WriteLn;
    Write('Copy previous arrival rates? (n/y): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, arrivals, DataFile);
    END
    ELSE
      FOR j := 1 TO NumberStations DO
      BEGIN
        Write('Enter station arrival rate for station ',j:2,': ');
        ReadLn(StationSourceRate[j])
      END;
    FOR j := 1 TO NumberStations DO
      BlockWrite(DataFile, StationSourceRate[j], SIZEOF(StationSourceRate[j]));
    WriteLn;

  (*StationMultiplex*)
    ClrScr;
    CenterText('Entry of traffic type/media type multiplex data');
    WriteLn;
    FOR j := 1 TO NumberStations DO
    BEGIN
      WriteLn('Entry of media multiplex vector for station ',j:3,'.');
      Write('Copy previous multiplex vector? (y/n): ');
      ReadLn(command);
      IF command = 'y' THEN
      BEGIN
        REPEAT
          Write('Enter previously defined station from which to copy: ');
          ReadLn(index)
        UNTIL index < j;
        FetchData(i, index, multiplex, DataFile)
      END
      ELSE
        FOR k := 1 TO NumberMediumTypes DO
        BEGIN
          Write('Enter fraction of medium ', k:2,' dedicated to this traffic: ');
          ReadLn(StationMultiplex[j, k])
        END;
        FOR k := 1 TO NumberMediumTypes DO
          BlockWrite(DataFile, StationMultiplex[j,k], SIZEOF(StationMultiplex[j,k]));
      WriteLn
    END(*FOR j...*);

  (*Topology*)
    ClrScr;
    CenterText('Entry of topology matrix for this traffic type');
    WriteLn;
    WriteLn('Note:  station "0" is the sink for all messages, so enter the ');
```

```
    WriteLn('       proportion of traffic terminating at node i for the ');
    WriteLn('       [i, 0] entry of the topology matrix.');
    WriteLn;
    Write('Copy previous topology matrix? (y/n): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, connectivity, DataFile)
    END
    ELSE
    BEGIN

      (*Initialize all transition probabilities to zero.*)

      FOR j := 1 TO NumberStations DO
        FOR k := 0 TO NumberStations DO Topology[j, k] := 0.0;
      WriteLn('Enter (origin, destination, probability), with data entries');
      WriteLn(' separated by spaces, followed by a <ENTER>.  To terminate ');
      WriteLn('the process, enter a probability of zero with any node pair.');
      WriteLn;
      REPEAT
        Write('Origin node, Destination Node, Probability --> ');
        ReadLn(j, k, sum);
        IF sum <> 0 THEN Topology[j, k] := sum
      UNTIL  sum = 0.0
    END(*IF command...ELSE...*);
    FOR j := 1 TO NumberStations DO
      FOR k := 0 TO NumberStations DO
        BlockWrite(DataFile, Topology[j,k], SIZEOF(Topology[j,k]));
    WriteLn
  END(*FOR i ...*);
  CLOSE(DataFile);

(*Data verification*)

  ClrScr;
  WriteLn;
  CenterText('DATA VERIFICATION');
  WriteLn;
  IF NOT Verify(FileName, FALSE) THEN
  BEGIN
    BEEP;
    TextAttr := TextAttr OR BlinkOn;
    CenterText('WARNING: this data must be edited before use.');
    TextAttr := TextAttr + BlinkOff
  ENd
  ELSE
    WriteLn('Data entered does not violate any consistency rule.');
  Write(': press any key to continue.');
  ReadLn
END(*CreateNetwork*);

PROCEDURE EditNetwork(FileName: STRING);
VAR i, j, EditChoice, index: INTEGER;
    command: CHAR;
```

```
    quit: BOOLEAN;
    value, sum: REAL;
    DataFile, TempFile: FILE;
    FileInfo: SearchRec;
    TempFileName, NewName: STRING;
    data: BYTE;
    PreLoop, LoopSize, InLoop, StartPoint: LONGINT;

BEGIN
  quit := FALSE;

(*Create a copy of the input data file on which to make editing changes.*)

  TempFileName := FileName + '.tmp';
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    Write('Cannot find the file to be edited: press any key to continue.');
    ReadLn;
    EXIT
  END;
  ASSIGN(DataFile, FileName);
  RESET(DataFile, 1);
  ASSIGN(TempFile, TempFileName);
  REWRITE(TempFile, 1);
  WHILE NOT EOF(DataFile) DO
  BEGIN
    BlockRead(DataFile, data, 1);
    BlockWrite(TempFile, data, 1)
  END(*WHILE NOT...*);
  GetGlobal(DataFile);          (*Get the required global parameters from the file.*)
  CLOSE(DataFile);

  (*Obtain essential "sizing" parameters for getting around in the file.*)

  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations + SQR(NumberStations) +
               NumberStations * NumberMediumTypes)*SIZEOF(REAL);

  (*Main edit menu follows.*)

  WriteLn;
  REPEAT
    WriteLn(' Edit functions are as follows:');
    WriteLn('   0. exit the edit function,');
    WriteLn('   1. modify media bandwidths,');
    WriteLn('   2. modify traffic type global arrival rates,');
    WriteLn('   3. modify traffic type traffic length,');
    WriteLn('   4. modify traffic type acknowledgement length,');
    WriteLn('   5. modify traffic type/medium type error rates');
    WriteLn('   6. modify traffic type station arrival rates,');
    WriteLn('   7. modify traffic type medium multiplex vector,');
    WriteLn('   8. modify traffic type station connectivity,');
    WriteLn;
    Write(' Enter integer corresponding to choice: ');
    ReadLn(EditChoice);
    CASE EditChoice OF
      0:        (*Exit procedure and saving edited file to disk.*)
```

```
BEGIN
  quit := TRUE;
  CLOSE(TempFile);
  WriteLn;
  Write(' Save as O(riginal, as N(ew, or E(xit without saving?: ');
  ReadLn(command);
  CASE command OF
    'o','O':
    BEGIN
      ERASE(DataFile);
      RENAME(TempFile, FileName)
    END(*CASE 'o'*);
    'n','N':
    BEGIN
      REPEAT
        Write(' Enter filename under which to store edited data: ');
        ReadLn(NewName);
        NewName := NewName + '.top';
        FindFirst(NewName, AnyFile, FileInfo);
        IF DOSError = 0 THEN
        BEGIN
          WriteLn;
          BEEP;
          TextAttr := TextAttr OR BlinkOn;
          CenterText('WARNING: File of that name already exists.');
          TextAttr := TextAttr AND BlinkOff;
          WriteLn;
          Write(' Press any key to continue.');
          ReadLn(command)
        END
        ELSE
        BEGIN
          RENAME(TempFile, NewName);
          WriteLn(' File ', NewName, 'stored to disk.');
        END
      UNTIL DOSError <> 0
    END;
    'e','E': ERASE(TempFile)
  END(*CASE command...*)
END(*CASE 0*);
1:                              (*Modify media bandwidths*)
BEGIN
  REPEAT
    WriteLn;
    Write(' Modify which medium bandwidth?: ');
    ReadLn(index)
  UNTIL index <= NumberMediumTypes;
  StartPoint := 2*SIZEOF(INTEGER)
                + (index - 1)*SIZEOF(REAL);
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn(' Existing value is ', value:8:2);
  Write(' Modify to what value?: ');
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  Blo      .e(TempFile, value, SIZEOF(value))
END(*CASE 1*);
2:                      (*Modify traffic type global arrival rate*)
BEGIN
```

```
  REPEAT
    WriteLn;
    Write(' Modify which traffic type global arrival rate?: ');
    ReadLn(index)
  UNTIL index <= NumberTrafficTypes;
  StartPoint := PreLoop + (index - 1)*LoopSize;
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn(' Existing value is ', value:8:2);
  Write(' Modify to what value?: ');
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 2*);
3:                  (*Modify traffic type traffic length*)
BEGIN
  REPEAT
    WriteLn;
    Write(' Modify which traffic type traffic length?: ');
    ReadLn(index)
  UNTIL index <= NumberTrafficTypes;
  StartPoint := PreLoop + (index - 1)*LoopSize + SIZEOF(REAL);
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn(' Existing value is ', value:8:2);
  Write(' Modify to what value?: ');
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 3*);
4:                  (*Modify traffic type acknowledgement length*)
BEGIN
  REPEAT
    WriteLn;
    Write(' Modify which traffic type acknowledgement length?: ');
    ReadLn(index)
  UNTIL index <= NumberTrafficTypes;
  StartPoint := PreLoop + (index - 1)*LoopSize + 2*SIZEOF(REAL);
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn(' Existing value is ', value:8:2);
  Write(' Modify to what value?: ');
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 4*);
5:          (*Modify traffic type/medium type error rates*)
BEGIN
  REPEAT
    WriteLn;
    Write(' Modify error rate for which traffic/medium pair?: ');
    ReadLn(index, i)
  UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberMediumTypes));
  StartPoint := PreLoop + (index - 1)*LoopSize +
                (2 + i)*SIZEOF(REAL);
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn('Existing value is ', value:6:4);
  Write('Modify to what value?: ');
```

```
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 5*);
6:              (*Modify station source rates for traffic type*)
BEGIN
  REPEAT
    WriteLn;
    Write(' C(hange station source rate, E(xit (c/e): ');
    ReadLn(command);
    CASE command OF
      'c', 'C':
      BEGIN
        REPEAT
          WriteLn;
          Write(' Modify source rates for which traffic type/');
          Write('station?: ');
          ReadLn(index, i)
        UNTIL ((index <= NumberTrafficTypes) AND (i <=NumberStations));
        StartPoint := PreLoop + (index - 1)*LoopSize
                    + (2 + i + NumberMediumTypes)*SIZEOF(REAL);
        SEEK(TempFile, StartPoint);
        BlockRead(TempFile, value, SIZEOF(value));
        WriteLn(' Existing value is ', value:8:2);
        Write(' Modify to what value?: ');
        ReadLn(value);
        SEEK(TempFile, StartPoint);
        BlockWrite(TempFile, value, SIZEOF(value))
      END(*CASE 'c'*);
      'e','E':
    END(*CASE command...*)
  UNTIL command = 'e'
END(*CASE 6*);
7:              (*Modify traffic type medium multiplex vector*)
BEGIN
  REPEAT
    WriteLn;
    Write(' C(hange a multiplex value, or E(xit (c/e): ');
    ReadLn(command);
    CASE command OF
      'c', 'C':
      BEGIN
        REPEAT
          WriteLn;
          Write(' Modify medium multiplex vector for which traffic');
          Write(' type/ station/ medium?: ');
          ReadLn(index, i, j)
        UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberStations)
              AND(j <= NumberMediumTypes));
        StartPoint := PreLoop + (index - 1)*LoopSize
                    + (3 + NumberMediumTypes + NumberStations
                    + (i - 1)*NumberMediumTypes + j - 1)*SIZEOF(REAL);
        SEEK(TempFile, StartPoint);
        BlockRead(TempFile, value, SIZEOF(value));
        WriteLn(' Existing value is ', value:8:2);
        Write(' Modify to what value?: ');
        ReadLn(value);
        SEEK(TempFile, StartPoint);
        BlockWrite(TempFile, value, SIZEOF(value))
```

```
          END(*CASE 'c'*);
          'e','E':
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE 7*);
    8:
    BEGIN
      REPEAT
        WriteLn;
        Write(' C(hange a transition probability, or E(xit (c/e): ');
        ReadLn(command);
        CASE command OF
          'c', 'C':
          BEGIN
            REPEAT
              WriteLn;
              Write('Modify topology for which traffic type/ origin');
              Write('/destination stations?: ');
              ReadLn(index)
            UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberStations)
                  AND (j <= NumberStations));
            StartPoint := PreLoop + (index - 1)*LoopSize
            + (3 + NumberMediumTypes + NumberStations*(NumberMediumTypes + 1)
            +(i - 1)*NumberStations + j)* SIZEOF(REAL);
            SEEK(TempFile, StartPoint);
            BlockRead(TempFile, value, SIZEOF(value));
            WriteLn(' Existing value is ', value:8:2);
            Write(' Modify to what value?: ');
            ReadLn(value);
            SEEK(TempFile, StartPoint);
            BlockWrite(TempFile, value, SIZEOF(value))
          END(*CASE 'c'*);
          'e','E':
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE 8*)
  END(*CASES*)
UNTIL quit
END(*EditNetwork*);

FUNCTION DisplayNetwork(TrafficIndex: INTEGER, FileName: STRING):BOOLEAN;
VAR i, j: INTEGER;
    PreLoop, LoopSize, StartPoint: LONGINT,
    DataFile: FILE;
    FileInfo: SearchRec;
    lst: TEXT;



BEGIN
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo),
  IF DOSError <> 0 THEN
  BEGIN
    DisplayNetwork := FALSE;
    EXIT
  END
  ELSE
  BEGIN
```

```
ASSIGN(DataFile, FileName);
RESET(DataFile, 1);
ASSIGN(lst, 'prn');
REWRITE(lst);
IF TrafficIndex = 0 THEN
BEGIN

(*print out the global information.*)

  GetGlobal(DataFile);
  WriteLn(lst, 'GLOBAL DATA FOR NETWORK DEFINITION IN ', FileName);
  WriteLn(lst);
  WriteLn(lst, 'Number of network nodes = ', NumberStations:3);
  WriteLn(lst, 'Number of medium types  = ', NumberMediumTypes:3);
  WriteLn(lst, 'Number of traffic types = ', NumberTrafficTypes:3);
  WriteLn(lst);
  WriteLn(lst, 'Media Bandwidths:');
  FOR i := 1 TO NumberMediumTypes DO
  BEGIN
    Write(lst, 'Bandwidth for medium ', i:3, ' = ');
    WriteLn(lst, MediumBandWidth[i]:8:2,' KBits/sec.')
  END;
  WriteLn(lst, FormFeed)
END
ELSE
BEGIN

(*Print out the specific information concerning a given traffic type.*)

  GetGlobal(DataFile);
  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations +
    SQR(NumberStations) + NumberStations*NumberMediumTypes)*SIZEOF(REAL);
  StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
  SEEK(DataFile, StartPoint);

(*global arrival rate, message length, acknowledgement length*)

  BlockRead(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
  BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
  BlockRead(DataFile, AckLength, SIZEOF(AckLength));
  Write(lst, 'INFORMATION FOR TRAFFIC TYPE ',TrafficIndex:3);
  WriteLn(lst ,' IN FILE ', FileName);
  WriteLn(lst);
  Write(lst, 'Traffic type global arrival rate    = ');
  WriteLn(lst, GlobalArrivalRate:9:2, ' messages/sec.');
  Write(lst, 'Traffic type message length         = ');
  WriteLn(lst, TrafficLength:8:1,' bits.');
  Write(lst, 'Traffic type acknowledgement length = ');
  WriteLn(lst, AckLength:8:1, ' bits.');
  WriteLn(lst);

(*missed message rates for all medium types.*)

  FetchData(TrafficIndex, 0, errors, DataFile);
  WriteLn(lst, 'Traffic type missed message rates for the medium types:');
  FOR i := 1 TO NumberMediumTypes DO
    WriteLn(lst, 'MMR for medium type ',i:3, ' = ',ErrorRates[i]:5:3);
  WriteLn(lst);
```

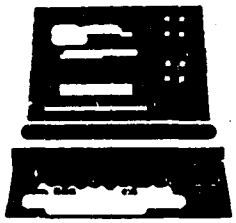
```
    (*source rates for traffic type at each station*)

      FetchData(TrafficIndex, 0, arrivals, DataFile);
      WriteLn(lst, 'Arrival rate for this traffic type at each station:');
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst,'Arrival rate at station ', i:3,' = ');
        WriteLn(lst,StationSourceRate[i]:5:4,' messages/sec.')
      END;
      Write(lst, FormFeed);

    (*station multiplex vectors*)
      FOR j := 1 TO NumberStations DO
        FetchData(TrafficIndex, j, multiplex, DataFile);
      WriteLn(lst, 'Multiplex vectors for traffic type ', TrafficIndex:3);
      WriteLn(lst);
      WriteLn(lst,' ':20,'Medium 1      Medium 2      Medium 3');
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst,'Station ',i:3, '      ':9);
        FOR j := 1 TO NumberMediumTypes DO
          Write(lst,StationMultiplex[i,j]:5:3, ' ':8);
        WriteLn(lst)
      END;
      Write(lst, FormFeed);

  (*traffic type topology matrix*)

      FetchData(TrafficIndex, 0, connectivity, DataFile);
      Write(lst, 'Routing transition probabilities for traffic type ');
      WriteLn(lst, TrafficIndex:3);
      WriteLn(lst, '(The "0-th" entry represents traffic absorption at station)');
      WriteLn(lst);
      Write(lst, ' ':5);
      FOR i := 0 TO NumberStations DO Write(lst, i:3, ' ':5);
      WriteLn(lst);
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst, i:2, '   ');
        FOR j := 0 TO NumberStations DO Write(lst, Topology[i,j]:5:3, ' ':3);
        WriteLn(lst)
      END(*FOR i...*);
      Write(lst, FormFeed)
    END(*IF TrafficIndex...ELSE...*);
    CLOSE(lst);
    CLOSE(DataFile);
    DisplayNetwork := TRUE
  END(*IF DOSError...ELSE...*)
END(*DisplayNetwork*);

FUNCTION DisplayThruputs(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;
VAR i: INTEGER;
    ThruputFile: FILE;
    StartPoint: LONGINT;
    value: REAL;
    FileInfo: SearchRec;
    lst: TEXT;
```

```
  BEGIN
    FileName := FileName + '.thp';
    FindFirst(FileName, AnyFile, FileInfo);
    IF DOSError <> 0 THEN
    BEGIN
      Write('Throughput file ', FileName,' could not be found:');
      WriteLn('press any key to continue.');
      ReadLn;
      DisplayThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      ASSIGN(lst, 'prn');
      REWRITE(lst);
      WriteLn(lst, 'THROUGHPUT DATA FOR TRAFFIC TYPE ',TrafficIndex:3);
      WriteLn(lst);
      ASSIGN(ThruPutFile, FileName);
      RESET(ThruputFile, 1);
      BlockRead(ThruputFile, NumberStations, SIZEOF(NumberStations));
      StartPoint := 2*SIZEOF(INTEGER)
                  + (TrafficIndex - 1)*NumberStations*SIZEOF(REAL);
      SEEK(ThruputFile, StartPoint);
      FOR i := 1 TO NumberStations DO
      BEGIN
        BlockRead(ThruputFile, value, SIZEOF(value));
        WriteLn(lst, 'Station ', i:3, ' throughput = ',value:5:3)
      END;
      Write(lst, FormFeed);
      CLOSE(ThruputFile);
      CLOSE(lst);
      DisplayThruputs := TRUE
    END(*IF DOSError...ELSE...*);
  END(*DisplayThruputs*);

BEGIN

END(*Data_IO*).
```

```
UNIT NetComp;                    (*John R. Doner, 3 September 1989*)

(*This unit contains the necessary computional proccedures to determine the
  network relative throughputs, and all derived network performance measures
  associated with the AIRMICS Multimedia Network Design Program (NetCalc).*)


INTERFACE

USES DOS, Data_IO;


(**************************************************************************
  The following procedure solves the required linear system of equations to
  obtain the network relative throughputs for a given traffic type.  Input
  to the procedure is the traffic type and the network file name containing
  the network data, and the output solution is then placed in the appropriate
  row of the StationThruput[ ] array (see Data_IO unit for definition.).
  FALSE is returned only if the equations were found to be non-solvable (i.e.,
  singular matrix).  The file containing network data should be closed at
  entry to this procedure, and will be closed at exit.  Note that this
  function does not store the computed throughputs to disk.
**************************************************************************)

  FUNCTION SolveThruputs(TrafficIndex: INTEGER; NetworkName: STRING): BOOLEAN;

IMPLEMENTATION

  FUNCTION SolveThruputs(TrafficIndex: INTEGER; NetworkName: STRING): BOOLEAN;
  VAR InvertArray: ARRAY[1..MaxNodes,1..MaxNodes + 1] OF REAL;
      TransposeCount, i,j, k: INTEGER;
      divisor, multiplier, temp: REAL;
      transpose: ARRAY[1..MaxNodes,0..1] OF INTEGER;
      FileInfo: SearchRec;
      DataFile: FILE;

  (*"InvertArray[]" holds the enhanced matrix while elementary row operations
    are performed.  "transpose" holds information on any row interchanges
    required during the upper triangularization process. *)

   (*The following function interchanges two rows of the InvertArray matrix if
     that is required to bring a non-zero into a diagonal position. The
     function returns FALSE only if there are no non-zero elements below the
     diagonal.*)

      FUNCTION Interchange(row: INTEGER)  BOOLEAN;
      VAR i, j, k: INTEGER;
          temp : REAL;

      BEGIN
        j := i + 1;
        WHILE ((j < NumberStations + 1) AND (InvertArray[j,i] = 0)) DO
          j := j + 1;
        IF j = NumberStations + 1 THEN
        BEGIN
          Interchange := FALSE;
          EXIT
        END;
```

```
      FOR k := i TO NumberStations + 1 DO
      BEGIN
        temp := InvertArray[i,k];
        InvertArray[i,k] := InvertArray[j,k];
        InvertArray[j,k] := temp
      END(*FOR k...*);
      TransposeCount := TransposeCount + 1;
      transpose[TransposeCount, 0] := i;
      transpose[TransposeCount, 1] := j;
      Interchange := TRUE
    END(*Interchange*);


BEGIN
(*First, open the file and fetch the relaevant data.*)

  NetworkName := NetworkName + '.top';
  FindFirst(NetworkName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    SolveThruputs := FALSE;
    EXIT
  END
  ELSE
  BEGIN
    ASSIGN(DataFile, NetworkName);
    RESET(DataFile, 1);

(*Solution for the thruoughputs is carried out by enhancing the coefficient
  matrix with the column of source rates for the nodes, and then transform-
  ing the coefficient matrix to upper triangular form. From this form, the
  unknowns (thruoughputs), can be iteratively determined from the last to
  the first  (Biblical method).*)

(*First load required data to InvertArray*)

    FetchData(TrafficIndex, 0, arrivals, DataFile);
    FOR i := 1 TO NumberStations DO
      InvertArray[i, NumberStations + 1] := StationSourceRate[i];
    FetchData(TrafficIndex, 0, connectivity, DataFile);
    FOR i := 1 TO NumberStations DO
      FOR j := 1 TO NumberStations DO
        InvertArray[i,j] := -Topology[j,i];
    CLOSE(DataFile)
  END(*IF DOSError...ELSE...*);

  FOR i := 1 TO NumberStations DO
    InvertArray[i,i] := InvertArray[i,i] + 1.0;
  TransposeCount := 0;

  (*Matrix is defined: ready to begin upper triangulation.*)

  FOR i := 1 TO NumberStations -1 DO
  BEGIN

  (*First, check that next diagonal element is non-zero, and perform a
    row transposition if necessary.*)

    IF InvertArray[i,i] = 0 THEN
```

```
        IF NOT Interchange(i) THEN
        BEGIN
          SolveThruputs := FALSE;
          EXIT
        END(*IF NOT ...*);
      divisor := InvertArray[i, i];

      (*Normalize i-th row so diagonal element = 1.*)

      FOR j := i TO NumberStations DO
        InvertArray[i,j] := InvertArray[i,j]/divisor;

      (*Now do subtraction of multiple of i-th row from each following row
        to zero out i-th column below diagonal.*)

      FOR j := i + 1 TO NumberStations DO
      BEGIN
        multiplier := InvertArray[j,j];
        FOR k := i TO NumberStations + 1 DO
          InvertArray[j, k] := InvertArray[j,k] - multiplier*InvertArray[i,k];
      END(*FOR j...*)
    END(*FOR i...*);

    (*Now solve iteratively backward, from last to first throughput, and
      place in StationThruput array.*)

    FOR i := NumberStations DOWNTO 1 DO
    BEGIN
      temp := InvertArray[i, NumberStations + 1];
      FOR j := i + 1 TO NumberStations DO
        temp := temp - InvertArray[i,j]*StationThruputs[TrafficIndex, j];
      StationThruputs[TrafficIndex, i] := temp/InvertArray[i,i]
    END(*FOR i...*);

    (*Need to perform interchange, if any, of solutions*)

     FOR i := 1 TO TransposeCount DO
     BEGIN
       j := transpose[i,0];
       k := transpose[i,1];
       temp := StationThruputs[TrafficIndex, j];
       StationThruputs[TrafficIndex, j] := StationThruputs[TrafficIndex, k];
       StationThruputs[TrafficIndex, k] := temp
     END(*FOR i...*);
    SolveThruputs := TRUE
  END(*SolveThruputs*);

END(*NetComp*).
```



USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

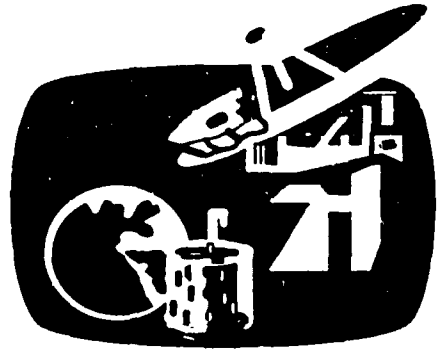
MULTIMEDIA NETWORK DESIGN

STUDY

(ASQB-GC-~~89~~ 90-002)

30 September 1989

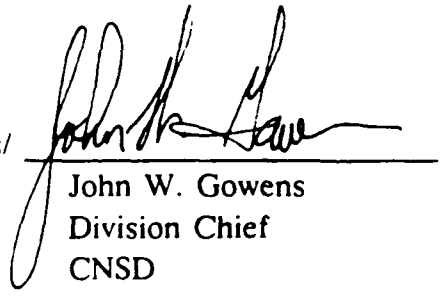
AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800



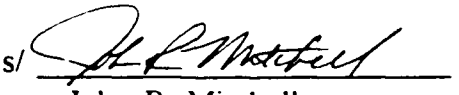
This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study with the work being done by the Harris Corporation. Its goal was to create a closed-form analytical queuing model for networks of queues. The Army's worldwide communication system has become a conglomeration of many systems such as DDN, DSN, AUTOVON, and wide or local area networks. The need for an efficient interconnection of these systems requires that systems be evaluated as a group as opposed to individual nodes. Therefore this research will provide a formal mathematical model specifically developed for the analysis of multimedia (e.g. coaxial cable, fiber optics, and twisted pair) networks. The primary output of the project is a PC/AT hosted program, MMDESIGN, which implements the formal mathematical model and provides a user interface for analyzing multi-media networks. The program is designed as an iterative analysis tool and can be used to derive steady state conditions for a network before attempting simulation analysis of the network to perform transient analysis. Pascal source code is provided in the report.

This research was performed under contract number DAK11-88-C-0052 for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ 

John W. Gowens
Division Chief
CNSD

s/ 

John R. Mitchell
Director
AIRMICS

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICAION AUTHORITY N/A	3. DISTRIBUTION/AVAILIBILTY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CLIN 0001 AA, SUN 0001 AB, CDRL A001/3	5. MONITORING ORGANIZATION REPORT NUMBER(S) ASQB-GC-002
6a. NAME OF PERFORMING ORGANIZATION Harris Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AIRMICS
6c. ADDRESS (City, State, and Zip Code) Melbourne, FL, 32902	7b. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS 8b. OFFICE SYMBOL (If applicable) ASQB-G	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAKF11-88-C-0052
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. P612783 / PROJECT NO. DY10 / TASK NO. 03-02-09 / WORK UNIT ACCESSION NO. DA 315 378
11. TITLE (Include Security Classification) Multimedia Network Design Study: First Year Final Report	
12 PERSONAL AUTHOR(S) Dr. John R. Doner	
13a TYPE OF REPORT Annual 13b. TIME COVERED FROM OCT 88 TO OCT 89	14. DATE OF REPORT (Year, Month, Day) 89, SEP, 30 15. PAGE COUNT 76
16 SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUBGROUP	Multimedia, Network, Queing, Modeling

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study. Briefly, the study goals for the three years of the effort are as follows. In the first year of the study, now completed, the goal was to create a closed-form analytical queuing model for networks of queues. The second year of the study, now beginning, will build on the effort of the first year by enhancing the utility of the network-of-queues model to provide automated optimization capabilities. The third year of the study will attempt to integrate the use of this tool with a quantitative approach to define type mission of a communications system, and evaluating in an exact way the effects of the communication system on mission-oriented (not communications-oriented) metrics.

In the first year, a careful literature search was completed to determine the scope and depth of the selected modeling technique and its applicability to large-scale complex communications networks. The general result of this search was that the techniques of closed-form analysis are applicable to certain important areas of network design, and in fact complement, rather than replace, simulation techniques. The issues which can be dealt with by closed-form analysis relate to the proportioning of traffic across the available media in the system.

20 DISTRIBUTIONN/AVAILIBILTY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT [ ] DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
CPT Timothy M. O'Hara	404/894-3136	ASQB-G

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

MULTIMEDIA NETWORK DESIGN STUDY

FIRST YEAR FINAL REPORT

SUBMITTED BY
DR. JOHN R. DONER, PRINCIPAL INVESTIGATOR

HARRIS CORPORATION
GOVERNMENT COMMUNICATIONS SYSTEMS DIVISION
P.O. BOX 91000
MELBOURNE, FL 32902

TABLE OF CONTENTS

1.0	EXECUTIVE SUMMARY	2
2.0	THE RATIONALE FOR CLOSED-FORM NETWORK QUEUEING ANALYSIS	4
2.1	Closed-Form Modeling as an Adjunct to Simulation	4
2.2	An Explanation of the Closed-Form Network-of-Queues Model	7
2.3	Computational Considerations for the Closed-Form Technique	14
2.3.1	The Computational Process in a closed Network	15
2.3.2	The Computational Process in an Open Network	16
3.0	CLOSED-FORM MODELING APPLIED TO MULTIMEDIA COMMUNICATION	17
3.1	The Multimedia Network Node	17
3.2	The Multimedia Network Composite Channel	18
3.3	The Error Process for the Composite Channel	19
3.4	Accounting fot Error Correction Traffic	20
3.4.1	Erroneous Traffic Effects at the Transmitting Node	21
3.4.2	Erroneous Traffic Effects at the Receiving Node	21
3.5	Computing Transmission Delays through the Network	22
4.0	INSTRUCTION MANUAL FOR THE MMDESIGN PROGRAM	24
4.1	Explanation of Program Inputs	24
4.2	Program Organization and Menus	27
4.2.1	User Interface Format	27
4.2.2	Intepretation of the Menus	28
4.2.2.1	The "N(ew" Command	28
4.2.2.2	The "C(reate" Command	29
4.2.2.3	The "E(dit" Command	30
4.2.2.4	The "H(ardcopy" Command	31
4.2.2.5	The "R(ecall" Command	32
4.2.2.6	The "T(hruput" Command	32
4.2.2.7	The "P(aths" Command	33
4.2.2.8	The "M(etrics" Command	34
4.2.2.9	The "Q(uit" Command	35
4.3	Summary and Further Directions	36
5.0	REFERENCES	37
	APPENDIX A -- MMDESIGN MENU NAVIGATION	38
	APPENDIX B -- OPEN NETWORK MATHEMATICS	39
	APPENDIX C -- MMDESIGN SOURCE CODE	43

1.0 EXECUTIVE SUMMARY

This document provides a report on the first year of the three-year AIRMICS Multimedia Network Design Study. Briefly, the study goals for the three years of the effort are as follows. In the first year of the study, now completed, the goal was to create a closed-form analytical queueing model for communication networks. The second year of the study, now beginning, will build on the effort of the first year by enhancing the utility of the network-of-queues model to provide automated optimization capabilities. The third year of the study will attempt to integrate the use of this tool with a quantitative approach to defining the mission of a communications system, and evaluating in an exact way the effects of the communications system on mission-oriented (not communications-oriented) metrics.

In the first year, a careful literature search was completed to determine the scope and depth of the selected modeling technique and its applicability to large-scale complex communications networks. The general result of this search was that the techniques of closed-form analysis are applicable to certain important areas of network design, and in fact complement, rather than replace, simulation techniques. Closed-form analysis of networks can only deal with steady-state equilibrium conditions in networks, such as the expected loading and delays in a network for which offered traffic, topology, and capacities have been allocated. This is quite different from the function that simulation performs, which is to study the consequences of specific scenarios in a network.

However, although simulation can yield very highly resolved insights into network behavior, it does so on a rather *ad hoc* basis: the simulator can test at most a very few of all possible communications scenarios which a network might be called upon to support. Closed-form analysis can provide globally applicable facts about a network, which may lead to early recognition of misallocation of capacity or potential for chronic overload. Using closed-form techniques, the network analyst can examine a wide range of network topologies and gain general insight into their suitability to meet mission requirements given the expected geographic and temporal variations in traffic load. These analyses will be much more rapidly executed than simulations, and the analyst can fairly rapidly determine which of a few candidate architectures appear in these general terms to support system requirements. Simulation studies can then proceed on this subset of architectures with the assurance that the candidate networks being simulated are at least close to the best answer satisfying system requirements.

Thus a well-executed design program for a large communications network should involve the interaction of simulation and closed-form analysis, with closed form analysis being used for a global estimation of the correctness of the network resource allocation, and simulation then following to test more specific aspects of the network design, such as routing strategies or survivability strategies. Such a design strategy will produce a more certain, and a less expensive answer, than will be obtained by simulation alone.

The intent of this study is to apply closed-form analysis to multimedia networks. A multimedia network will carry multiple types of traffic on multiple media. Each traffic type will be more or less suited for transmission on any given medium, depending on the medium bandwidth and error properties. Each traffic type may have certain essential constraints on its handling, related to timeliness and maximum permissible error acceptable for the traffic type. In a military

network designed for high survivability and maximum efficiency, the proper multiplexing of traffic types on the media can be an important factor in achieving such goals.

This requirement to multiplex traffic types across various media could be accomplished in several ways, such as allocating a specific proportion of each medium to each traffic type. The extent to which each traffic type would meet its timeliness and accuracy constraints would then be a function of that allocation. Since not every traffic type can at all times travel by the medium that is "best" for it, a process of compromise is necessary. It is precisely the determination of such an "optimum" compromise that is well served by the tools of closed-form network modelling.

The body of this document provides a formal mathematical model of multimedia traffic flow which encompasses the concepts of multiple traffic types, and multiple media types. The interaction of channel error processes with the traffic types is accurately captured, so that the multiplexing of traffic types on media types can be usefully analyzed. The primary output of this first year of effort is a computer program, called the MMDESIGN program which addresses the above analysis concerns. This program prompts the network analyst for a network design (i.e., topology, media, traffic types, routing, etc.), and then makes available the expected path delays associated with any traffic type over any path, or collection of paths. The program is designed as an iterative analysis tool: that is, the manner in which data is gathered, stored, and edited facilitates the analyst's normal activities in pursuit of the optimization of network performance relative to traffic multiplexing concerns.

The MMDESIGN program is hosted on an IBM PC/AT (or equivalent) and is written in Turbo Pascal, which is very widely available and well known to IBM PC programmers. This choice of computer limits the size of the network that can be handled, due to memory constraints, but the computer serves as a good base for wide dissemination of the program. The code is largely machine independent, and so could easily be ported onto a larger machine.

The remaining sections of this document provide an explanation of the closed-form modeling paradigm, its application to multimedia networks, and its implementation in the MMDESIGN program. Section 2 provides a rationale for and a description of the closed-form network-of-queues modeling paradigm. Section 3 explains the manner in which multimedia communications can be cast into that mold. Section 4 is a self-contained manual for the use of the MMDESIGN program. The appendices to the document provide complete detail and documentation of for both the mathematics and the computer code used in the program.

MMDESIGN will be extended in the second year to consider inclusion of optimization techniques within the program, such as optimization of capacity assignment and routing. The design of truly integrated multimedia communications systems is in a practical sense still in its infancy. It is hoped that this study will provide valuable tools to the operational network design community which must actually come to grips with the next generation of communications systems.

2.0 THE RATIONALE FOR CLOSED-FORM NETWORK QUEUEING ANALYSIS

In this section, we will introduce the rationale for the closed-form communications network modeling paradigm which has been the subject of this study. We will also provide a rather comprehensive survey of the applicability of the closed-form technique. This survey is not intended to be mathematically detailed, but does introduce terminology and concepts for the purpose of providing the reader with a comprehension of the general strengths and weaknesses of the technique.

2.1 Closed-Form Modeling as an Adjunct to Simulation

Modern military communications systems are rapidly evolving to take advantage of increasingly versatile communications technology. Procurement planning for the near-term future calls for increasingly survivable communications architectures which rely on an eclectic suite of communications assets. A major interest of all the military services is to fully integrate the use of multiple communications media into a single communications capability, the operation of which requires as little user management and intervention as possible. Such a system is expected to autonomously determine and ameliorate conditions detrimental to the expeditious flow of information, thereby creating a whole that functions better than the sum of its parts.

This idealized concept requires considerable innovation and experiment in the discipline of network control. Systems will in general comprise larger collections of assets, deal with a greater variety of traffic types, and be expected to handle larger volumes of traffic. Such designs will tax not only the traditional communications network design methods, but also the existing network design tools by which such designs are refined from concept to implementation.

The present study is a three-year effort funded by USA AIRMICS to consider the emerging design problems discussed immediately above. The intent of this study is to consider several concepts related to the design tools available to the network design community, and to create tools complementary to those now existing which will be specifically helpful in addressing the new multi-media, large-scale network designs of the near future.

This document reports on the results of the first year's effort, the establishment of a closed-form network-of-queues approach to modeling communications systems. Since most communications system design efforts rely heavily on simulation of the network, the rationale for creating a closed-form analytical model of network communications needs explanation. Simulation is, in essence, a process of determining single-point estimates of a very complex function. The inputs to the simulation constitute the independent "variable" (normally a multi-dimensional vector) of the function, and the outputs from the simulation constitute the value (again, normally a multi-dimensional vector) of the function. The user of the simulation selects an input value, runs the simulation, and obtains an output value.

Based on an iterative sequence of such simulation runs, together with modification of the simulation and/or its parameters, many major aspects of the network design may be determined. However, such simulation efforts generally constitute a sort of intuitive optimization process where the output of each simulation step guides the designer toward changes in the network design which will (it is hoped) provide better performance in the next simulation. In effect, the simulation user is attempting to discover the shape of a surface in a many-dimensional space by examining a sequence of "points" on that surface, and then selecting another value for the input argument to the function (i.e., simulation), which will move the output "point" uphill. This process is somewhat like that pictured in Figure 2 - 1 below. Since the simulator can only guess at the shape of the surface near the set of "points" already collected, it is never possible to assure that a network design based on simulation has actually achieved optimum performance within the design constraints.

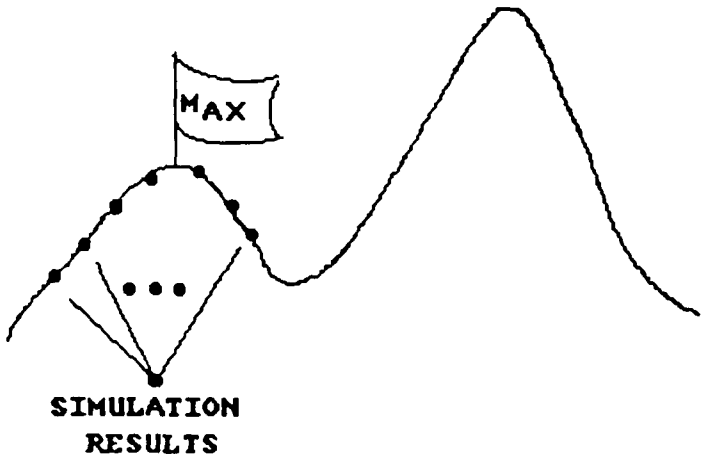


FIGURE 2 - 1: Simulation "Mountaineering"

Of course, the simulator, by examining as many "points" as possible on the simulation surface, can reduce the likelihood that there might be a better solution "near" the final chosen network design. But simulation as applied to modern and future military network designs tends to be expensive, and the number of events and communications paths to be simulated tends to grow combinatorially with the number of network nodes. As future networks move toward integration of all available media, it will be impossible to decompose the systems into multiple small networks, and so the need to handle very large numbers of events and assets in a single simulation will grow.

Having thus examined the conceptual and practical limitations of simulation, in what way can a closed-form, network-of-queues model contribute to network design? First, it should be admitted that such closed-form models are almost always bypassed in network design studies in favor of simulation. The reason for this is that a closed-form model can only model the functioning of a network operating in steady-state performance. Clearly, designers of military communications systems are very interested in guaging the response of the

network to many types of transient effects, and so cannot rely solely on steady-state network performance to select the parameters of their system. However, when only simulation is used, the steady-state performance can only be determined by long simulation runs, and each small change of input conditions may require another simulation run to obtain the changed steady-state.

Steady-state quantification within a closed-form network-of-queues paradigm is a much more convenient process. It is safe to say that a network designer could determine a great number of steady-state network solutions in the time required to determine a single steady-state solution by simulation. Moreover, since the closed-form model is analytical (i.e., expressed in terms of equations) in nature, there is the possibility of applying optimization procedures to the equations that describe the model, thereby obtaining a network design optimized in some respects directly from a single computer run of the model. Furthermore, this result may be a true optimum, rather than just a local maximum, as is more likely to happen when the optimization process proceeds essentially intuitively by means of simulation.

It is precisely this observation that justifies the use of a closed-form steady-state model of the system not in lieu of, but as an important adjunct to simulation. The designer then has an appropriate tool (simulation) by which to study system transient response, but can also more accurately "size" the network in terms of total assets required to meet the traffic demand of the system. An appropriate network design trajectory then uses the closed-form model to gain a global understanding of the network topology and link capacities required to efficiently meet the overall capacity demand at all points in the system. From this overview, simulation effort commences to resolve the more specific concerns of protocol development and allocation of resources at the individual nodes.

This interplay of simulation and closed-form analysis can also be used to advantage at later stages of system analysis. When network performance is to be analyzed over a range of scenarios including hostile actions against the system, simulations are usually done to demonstrate the manner in which the system recovers from loss of assets. Again, simulation is used here to examine system transient response as it moves from one steady-state (i.e., fully capable) to another. However, as was the case for design of the full network, if simulation is the only tool applied to this situation , then not all available information is being used. I.e., if the network transits from a fully capable state to an impaired state, then each of these states, through appropriate allocation of assets, can achieve some optimal performance relative to the network mission. If the optimum configuration in both circumstances is known, then simulation effort can be directed at fine-tuning network algorithms so as to obtain the transient response which moves the system toward its new steady-state in the most effective manner.

Summarizing the above points, simulation by itself is usually not adequate for a determination of the true optima possible for a network. As communications networks come to include ever larger suites of equipment, all integrated to serve as a single system, simulation alone will become less able to determine the best use of all the system assets, and the use of closed-form network-of-queues modeling will provide a valuable adjunct to the simulation effort. It does not provide the ability to examine specific protocols and routing techniques, as

simulation does, but it does permit the possibility of better global optimization and distribution of assets. A large-scale network design effort will generally be better served by using closed-form and simulation techniques together.

2.2 An Explanation of the Closed-Form Network-of-Queues Model

The terminology "closed-form" usually refers to technical results expressed in an equational format, such that all input parameters are independent variables of the equations, and the desired outputs are obtained by direct solution of the equations. When the technology in question is too complex to admit of such representations, it is usually necessary to rely on some sort of iterative solution procedure based directly on a mechanical characterization of the system interdependencies and how they serve to dynamically alter the system state. Solutions of problems in finite element analysis, in iterated differential (or difference) equations, and in simulation of system interactions all represent this genre of problem solution.

As was stated in the last section, most iterative solution techniques provide answers where no closed-form technique is available. Closed-form techniques, when available, have the intrinsic advantage of permitting mathematical manipulation and analysis of the equations involved, thereby providing the application of the great range of powerful mathematical optimization techniques available in the rich literature of optimization theory (see, e.g., [1] and [2]).

In the specific technology of queueing theory, the usual situation is that closed-form queueing techniques are confined to the characterization of single queues, or perhaps parallel queues with parallel servers all operating at a single service location. Most elementary queueing theory texts limit the development of the subject to such situations, and do not endeavor to discuss networks of queues at all. However, there is an extensive literature on this subject which has been evolving for about three decades, and has only recently found its way into textbooks and large-scale applications.

Some of the earliest work toward extending queueing theory to networks of queues was done by R.R.P. Jackson (see [3]) in 1954. The main supposition which allows queueing effects at one node to be visited in an analytically tractable way upon the activities at other nodes is the assumption that the future behavior of the system as a whole is dependent on past behavior only in terms of the current customer backlogs in the system nodes. Making this assumption tended to place certain limits on the variety of queueing protocols which could be modeled, and some of these limits will be discussed below. Substantial progress was made after these early papers by various scholars, and the type of network in which the future of the entire network system could be determined solely from the present state of the queues at all nodes became known as "product-form" networks. A very significant paper in this area was published by four authors (see [4]), Baskett, Chandy, Muntz, and Palacios. The paper pulled together some of the disparate results in the area, and also extended the product-form queueing model to a wide and coherently described set of conditions. Lemoine gives an excellent survey of the technology in [5], and Lavenberg discusses the practical computational aspects of the technique in Chapter 3 of his excellent textbook, *Computer Performance Modeling Handbook*, [6].

The exposition to be given in this section will follow the form, but not necessarily the notation, of the BCMP paper. Also, the exposition in this section will try to provide the reader with a sense of the scope to which the product-form network theory can be applied, while leaving explicit mathematical development to Appendix B.

It will, however, be necessary to introduce some symbolic notation. First, suppose that

N = number of service centers (nodes) in the system, and

R = number of classes of traffic.

These classes of traffic are distinct from each other in that they can follow distinct routing schemes, and have distinct service time and arrival rate distributions at the nodes. Routing is defined probabilistically in such a network by

P[(i,r), (j,s)] = probability that traffic of class r, at node i will transit to traffic of class s, node j.

These are called routing transition probabilities, and are normally considered to be expressed as an NRxNR matrix which has great convenience for computational purposes. There is a simplification of notation possible here which does not affect the applicability of the above equation, and that is to regard customers of the same class at different nodes as being designated by different class indices. This evades the need to consider a matrix indexed by pairs, so we can then write the matrix P[ ] as

P[i, j] = probability that a customer of class i will transit to class j. (Equation 2 - 1)

Thus, routing permits traffic to move between traffic classes and service nodes in a single transition. However, the fact that routing is probabilistic means that no particular traffic entity travels any specific route through the system: the routing paradigm permits statements about average channel utilization and expected traffic flows along links to be made, but does not support a completely detailed routing plan. This is the reason that closed-form models cannot replace simulation for the purpose of examining the detailed effects of routing algorithms.

In most queueing situations such as this, certain traffic classes travel in closed routing chains. I.e., not all possible transitions between traffic classes may occur, and not all traffic classes visit all nodes. In typical mathematical fashion, the analyst can seize upon this opportunity to study the entire problem as a sequence of sub-problems. Thus, without too careful a formal exposition, we define a routing chain as consisting of a subset of traffic classes and a subset of nodes such that the traffic classes only transit among themselves, and the traffic classes involved only circulate among the nodes in the subset. This does not say that other traffic classes do not also pass through these nodes: also, if multiple routing chains do pass through the same queues, the overall state of that queue can be expressed from the analysis of the separate routing chains. In this way, the

analysis of the entire system can be done by analyzing the separate routing chains, and then extending these results to the interactions of the routing chains in order to assess the complete network system. Thus we will first describe the terminology and results associated with a single routing chain.

A routing chain is called closed if the total count of customers in the chain remains constant over time. Where this is not the case, the routing chain is open. Closed systems are often used to model the processing interactions and delays in a computer time-sharing system, where some constant number of tasks are being "simultaneously" served by several types of system servers (e.g., disk access, printer access, terminal access, CPU access), and the same number of jobs shuttle from one service to another. Computer system designers can judge the expectations of processing delay and resource utilization for a steady-state load of a given constant number of on-line time-shared jobs by formulating such a closed network of queues. Open routing chains may have arriving and departing customers, and thus one does not *a priori* know what total number of customers will be in the system at any moment.

We have progressed far enough now to state the most important computational advantage of product-form networks. The "state" of a product form network at any moment is (by earlier assumptions) given entirely in terms of the lengths and compositions (in terms of numbers of customers of each customer type) of the queues at the various nodes. I.e., the basic tenet upon which the product-form of network analysis rests is that the future of the network depends only on the present condition of the queues at all the service centers. Thus, the state of the network is equivalent to the collective states of the nodes, and the state of any node is entirely determined by the numbers of each class of customer in the queue of that node. Thus, if S_p is an R-dimensional vector the components of which represent the numbers of customers of each type in queue at node p, then

$$(S_1, S_2, \ldots, S_N)$$

represents the state of the entire network. If Pr{.} represents the probability of an event described within the brackets, then we may state that

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\} \qquad \text{(Equation 2 - 2).}$$

This equation states that the probability of the global state of the system, as represented by all of the individual queue vectors S_i is equal to the product of the probabilities of the respective queueing situations arising individually at the separate nodes. I.e., there are no intermodal effects and dependencies to affect the analysis, and we may divide and conquer the analysis problem by focusing on the behavior of a single node. The main result of the analysis is to provide closed-form expressions for the values $Pr\{S_i\}$, from which nodal response time, link utilizations, path delays, and other standard queueing metrics can be derived.

Having reduced the problem to examining single nodes in a single routing chain, we now taxonomize the types of queueing disciplines that may be treated by this analysis. First, all arrivals to the network have the Poisson distribution, and separate traffic classes may have separate arrival rates. The arrival rate for a given traffic class is usually defined globally as a single Poisson process which is

then subdivided among the nodes by a constant probability distribution, but it is equivalent to consider separate Poisson arrival rates at the individual nodes which sum to the global rate (the former conceptualization is more practical in terms of the mathematics of the model). The Poisson arrival rate to the system for any customer class need not be constant: it can be an arbitrary function dependent upon the number of customers of that class already in the system.

Of course, in a closed system, all arrival rates are taken to be zero. In an open system, if there are arrivals, then there must also be departures; the departure process is normally formulated in terms of a single "sink" for each traffic type, with traffic of that type being routed to the sink from each node via a routing transition probability. (This is logistically equivalent to some means of allowing customers to leave the system at individual nodes.) Thus the departure of traffic from the system is easily encompassed in the routing transition probability structure described by equation 2 - 1.

The final major element of the model has to do with allowable queueing disciplines and service time distributions. Product form assumptions can be realized for the following four general types of queueing disciplines. (Some other queueing disciplines have been shown to yield product form networks, but only for specialized topologies: e.g., see [7].)

The first queueing protocol permitted is the very common first-in, first-out (FIFO) queue. This is the most commonly encountered queue, where customers are placed in the queue in the order of their arrival, and are served in order of their arrival. Thus a newly arrived customer waits behind all previously arrived customers, and is served only after all previously arrived customers have completed service. Such a queue is illustrated in Figure 2 - 2. If a service node has a FIFO queue, then all customers which pass through that node are subject to the same exponential service time distribution.

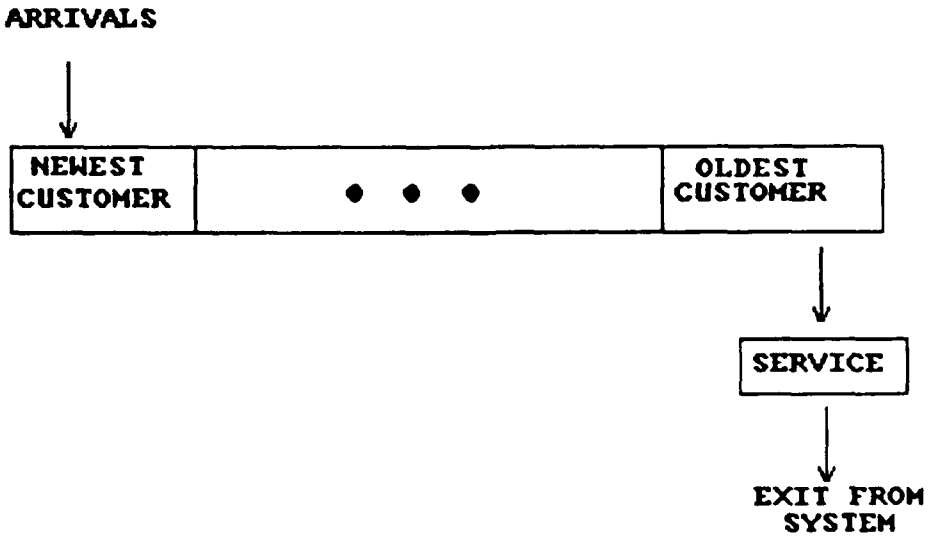


Figure 2 - 2 -- The First-In, First-Out (FIFO) Queue

The second type of queueing protocol possible at a service node is the processor-sharing (PS) mode of queueing. In this type of queueing, all customers at the node simultaneously share the server. Thus, each newly arriving customer receives immediate service, but the server accomplishes this instantaneous response only by slowing down the service rate at which all already present customers are served. Thus if the overall service rate is μ, then when K customers are present at the node each customer is served at rate μ/k. This type of queueing occurs in time-shared computer systems, and is illustrated in Figure 2 - 3.

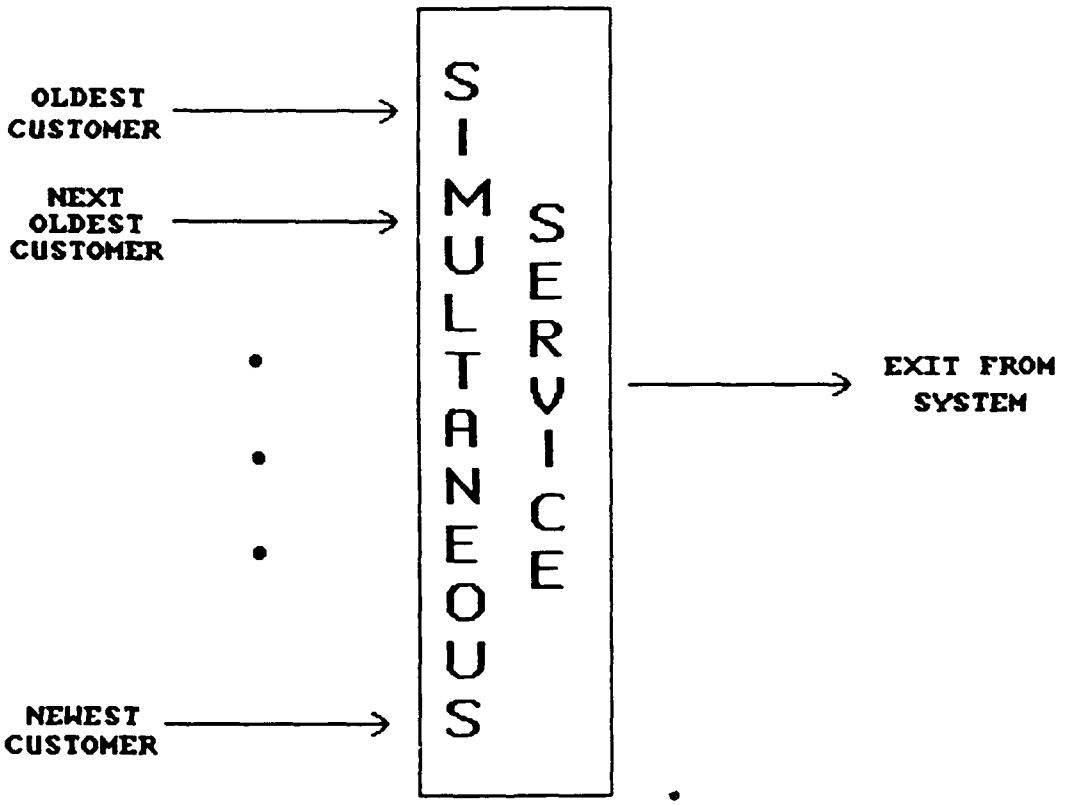


Figure 2 - 3: The Processor-Shared Queueing Discipline

Customers at a PS node can have distinct service rates, depending on their customer class. The service rate distribution can be any probability distribution with a rational Laplace transform. In effect, this means that the service operation can be thought of as consisting of a sequence of exponential service operations, each with independently determined mean, and with the possibility of the customer exiting the service after any one of the service steps. This type of service is depicted in Figure 2 - 4.

If a service operation of this type for customer class i contains only a single service operation, the service time distribution will be exponential. In this case, the only parameter of the service rate is μ_i, and $1/\mu_i$, the mean service time, can then be an arbitrary function of the number of customers of type i at the service center. Thus, μ_i can be expressed as a discrete function

$\mu_i(1), \mu_i(2), \ldots, \mu_i(j), \ldots$

where the argument j represents the number of customers of type i at the node.

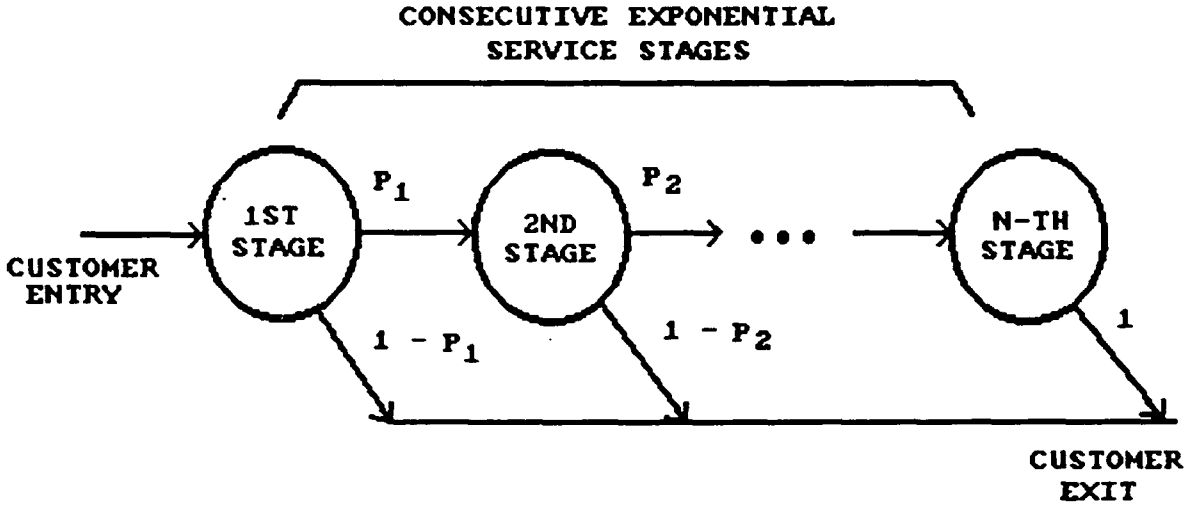


Figure 2 - 4 -- Schematic of Laplacian Distribution

The third type of queueing permitted at a service node is called infinite server (IS) queueing. In this case, the node always has more servers available than there are customers present in the node. The delay through such a node is thus strictly the service time delay associated with the customer class. The limitations on the service time distribution in this case are identical to those for the PS node described above. This node is illustrated in Figure 2 - 5.

IS nodes do not exist in real-world queueing systems, but they are useful when a single stage of delay is desired for a customer, with the delay being independent of other customer congestion in the system. E.g., in a communications system, a message which has waited behind other messages for access to a channel may, at the beginning of its actual transmission, wait for a channel access slot to become available in a round robin token-passing arrangement. This final short delay before transmission has nothing to do with other traffic in the system, and can be conveniently modeled by the IS queue.

The final form of queueing which is possible for product-form service nodes is last-come, first-served (LCFS) queueing. In this type of queueing, any newly arrived customer will preempt a customer already in service, and service for the preempted customer will then be suspended until the new customer has completed service. When a customer completes service, the most recently preempted customer resumes service. This type of service is evident, for example, in computer operating systems, where an interrupt to a processor causes the processor to suspend service to the present task and turn attention to servicing the interrupt. If another interrupt occurs during the service of the first

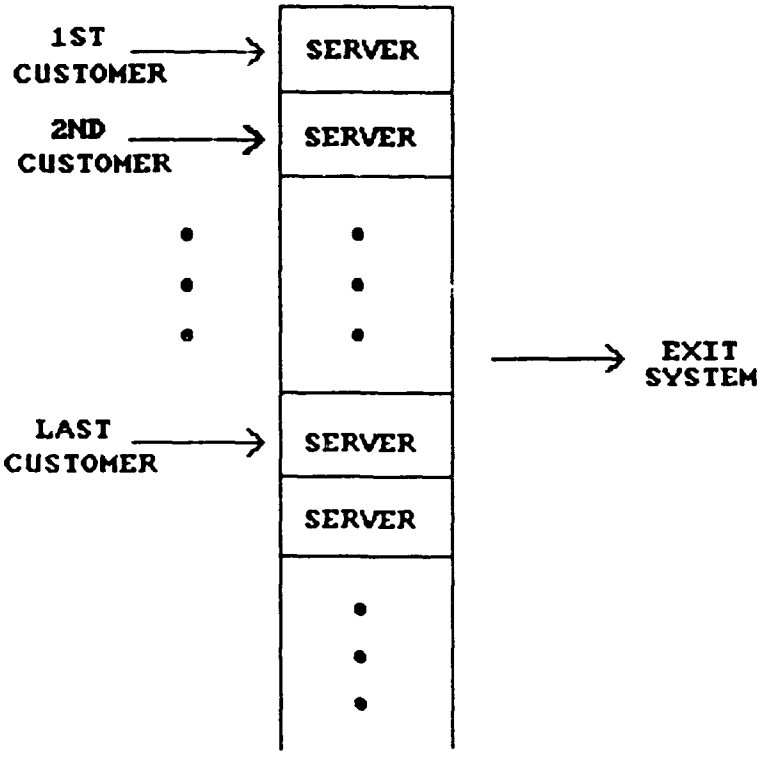


Figure 2 - 5: Infinite Server Queueing

interrupt, then the latter interrupt preempts the present interrupt. (Of course, many computer systems now have a prioritized interrupt structure, so that strict LCFS queueing would not apply.) LCFS queueing is illustrated Figure 2 -6.

The service time distributions possible for LCFS queueing are identical to those for PS and IS queueing. However, there is an added subtlety here, because preemptive queueing processes may or may not conserve the work already done on a customer. In the case of product-form LCFS queueing, the preemption is somewhere between conserving and non-conserving. Specifically, if the preempted customer has a multiple-stage service time distribution (see Figure 2 --3), then the customer is returned to service at the beginning of the stage in which preemption took place I e., the service already performed in earlier stages is preserved at preemption, but the service already performed at the current stage is lost. A final observation is that, for one-stage service time distributions, it follows that the preemption is non-conserving.

This effectively completes the description of the general topological and queueing models available for product form networks of queues. A given service center in such a network may apply any of the above four forms of queueing, and the flow of traffic, although stochastic, permits a customer of any class at any node to transit to any class and any node It should be pointed out that

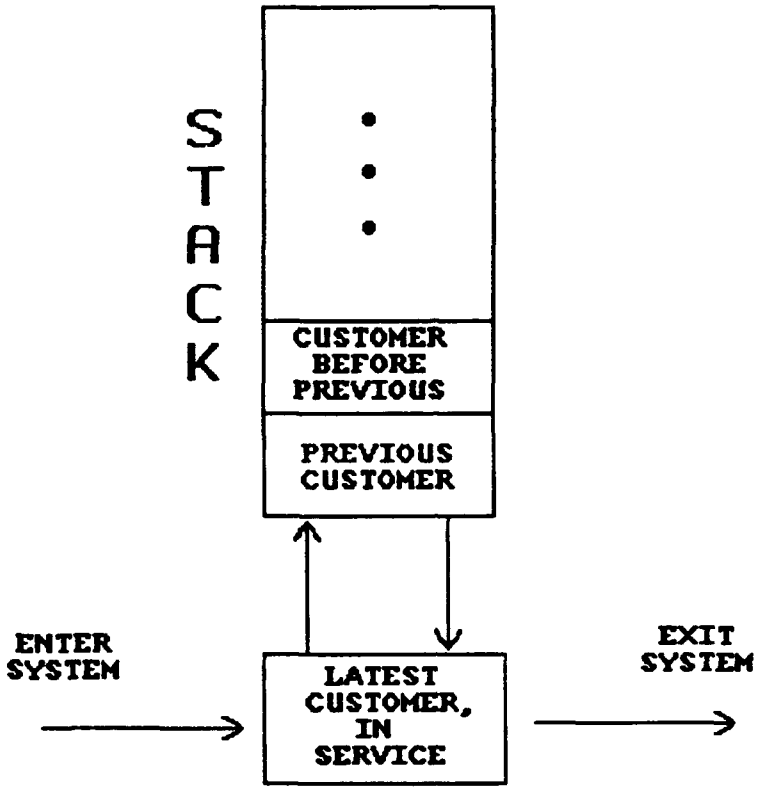


Figure 2 - 6: Last-Come, First-Served (LCFS) Queueing

customers can be effectively "deterministically" routed in this system by setting the appropriate transition probabilities equal to one. Also, it should be mentioned that the "service node" as described here is a mathematical artifact in terms of which the product form theory has been developed. In practice, the concept of a service node may involve several steps of processing of traffic in series and parallel combinations of the product-form service nodes. Thus, an actual communications network node may not be realistically reducible to a single node of one of the above types, but the delays and actions of the communications node may be adequately expressible in terms of a combination of the product-form nodes.

For example, suppose that we have a multimedia node at which messages of different types arrive. This node may be both processor-limited and bandwidth-limited, so that the nodal processing slows in proportion to the traffic in the node, and the traffic also backs up in queue at the output, waiting for service on the various media channels available . Such a node might best be modeled by a PS queue, followed by a FIFO queue.

2.3 Computational Considerations for the Closed-Form Technique

The present discussion would be incomplete without some reference to the practical computational complexity of applying the product-form network-of-

queues model. The difficulty associated with practical computation depends on whether the network is an open or closed network. The reason for this is that a fundamental quantity by which expected nodal loading, and thus all derived performance measures, are guaged is the "relative throughput" of each traffic class entering a node. The relative throughputs of a given customer class at a node is related to the routing in the system from all other nodes by the equation

$$y_d = P[0,d] + \sum_{c \in C} y_c P[c,d] \qquad \text{(Equation 2 - 3)}$$

where the y_d are the relative throughputs (one for each customer class/node pair in the network), and P[0,d] represents the arrivals to the class/node of new customers of the class d. The summation is taken over C, the set of all customer classes defined for the network. (The meaning of customer class follows the convention that each node/class combination is a distinct class, as was introduced in connection with Equation 2 - 1.) The situation now becomes quite different for open and closed networks, so these two situations will be treated separately in the following subsections.

2.3.1 The Computational Process in a Closed Network

The greatest computational difficulty arises from the fact that in a closed network, the quantities P[0,d] are all zero because there are no new arrivals to the system. The set of equations 2 - 3, with the quantities P[0,d] all set to zero, are linearly dependent because the coefficient matrix of the equations is Markovian and therefore has the sum of all columns equal to a vector of 1's. The result is that the relative throughputs, when solved for closed-form networks, are determined up to an unknown factor, i.e., the true class throughputs in the system constitute a vector which is a scalar multiple of the relative throughputs. The relationship is thus

$$(Y_1, Y_2, \ldots, Y_L) = (\alpha y_1, \alpha y_2, \ldots, \alpha y_L) \qquad \text{(Equation 2 - 4)}$$

where

Y_i = the absolute throughput for class i, and
α = the unknown constant relating absolute and relative throughputs.

The unknown value α is called the normalization constant. The process of determining α constitutes the bulk of the computational effort required to make the algorithm computationally feasible.

The unknown scalar α can be determined using the relative throughputs, by summing the values of the distribution

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\}$$

(see Equation 2 - 2) which theoretically must add to one. When the factors on the right are individually computed, using the relative throughputs, and the probability density in equation 2 - 2 is summed over all possible values of the nodal state vectors $S_1, S_2, \ldots, S_N$, the resulting sum will be in error by exactly the factor α.

Although straightforward enough in concept, this summation can be very large, since it effectively involves enumerating all possible combinations of queue backlogs jointly considered over all nodes. (However, since there are a fixed number of customers in a closed network, the computation is not infinite.)

A great many of the papers in this field have been devoted to decreasing the computational complexity associated with this step. Three main techniques are prominent, each of which may be favored under certain circumstances (see [6], pp. 145 - 151 for an excellent exposition of these techniques). The three techniques are known as recursion, mean value analysis, and the local balance algorithm for normalizing constants. Two very recent major algorithmic approaches to the computation of normalization constants are the RECAL algorithm described in [8] and the DAC algorithm discussed in [9].

2.3.2 Computational Process in an Open Network

The great difficulty involved in computing normalization constants for closed networks disappears completely for open networks. That is because in Equation 2 - 3, the arrival rates are non-zero, and so the linear equations are no longer singular. Consequently, the main computational difficulty is simply to solve the linear equations represented by Equation 2 - 3. This effectively can be done by any of a large number of efficient matrix inversion techniques. Generally, the major limitation of the technique for the open network is the size of the matrix of routing transition probabilities. Bear in mind, however, that where the customers in the system break into a number of disjoint routing chains, the full set of equations represented by Equation 2 - 3 also decomposes into smaller sets of disjoint independent equation sets. In the types of network applications that we will pursue, we will normally be dealing with open networks, decomposable into a number of disjoint routing chains (in fact, one chain for each traffic type). Therefore, the computational effort described in this section need not reflect the full complexity of the network in a single large set of linear equations.

3.0 CLOSED-FORM MODELING APPLIED TO MULTIMEDIA COMMUNICATION

The previous section of this document gave the reader a survey of the applicability of the closed-form, network-of-queues modeling techniques, and of the computational complexities involved. Based on that survey, the present section will examine the multimedia communications network, and provide a modeling paradigm for that network which utilizes product-form network-of-queues techniques.

Before launching into this development, it will be worthwhile to clearly state that the purpose of this modeling effort is to provide a means of studying the multiplexing of traffic types on the media types in a multimedia network in such a way that a fully integrated multimedia network (as opposed to a collection of separate single-medium networks sharing common nodes) results, with optimum use of the media relative to the characteristics of the traffic types.

A final point that should be mentioned is that the model developed here is in effect a prototype, and is kept fairly simple for that reason. It treats the network in a rather simplified form, and is limited in scope to the examination of traffic loading issues in the system. Since this is the first year's output for a three-year study, the prototype will be expanded in many ways to serve a more detailed set of issues in multimedia networks. To what extent this prototype can be expanded in scope will depend on further experience gained with the prototype and with the computational efficiency of the prototype. This can only be ascertained after the prototype has been exercised over some range of examples in the second year of the study.

This section includes some mathematical development which is essential to understanding how the closed-form modeling paradigm has been adapted to multimedia communications networks. The mathematics developed here is not part of the general mathematics of product-form networks of queues; it was developed explicitly to correctly represent multimedia communications concepts in terms of the product-form model. A user of the MMDESIGN program must understand the concepts in this section in order to intelligently apply the MMDESIGN program to design issues.

3.1 The Multimedia Network Node

A multimedia network node will be characterized for our purposes as a node which accepts traffic from other nodes, and on various input links, and can then pass traffic from itself to other nodes along various links. The links entering and leaving the node can be supported by various media and modulation types, and the traffic entering and leaving the node can be of different types. The main distinctions which will be drawn between media in this model will be the bandwidths which each medium makes available for traffic, and the signal degradation properties of the medium/modulation pair as it effects each traffic type's error rate. The main distinction between traffic types that will be drawn will be the differing error rates induced by the various media, and the mechanisms by which erroneous traffic is handled

Internal to the network node, we must identify a queueing discipline which is compatible with those supported by the product-form model, and is also compatible with our desire to model traffic flow realistically in a communications system. Of the four queueing disciplines available (see Section 2.2), the FIFO discipline is the most reasonable match to ordinary traffic queueing. I.e., traffic leaving a node may need to be queued because the bandwidth available on some medium is less than required to immediately service the current traffic offered.

However, an irritating complication arises if we simply try to model each medium leaving a node as a single FIFO queue, and that is that the FIFO queueing discipline for product-form networks constrains all customers entering the queue to have the same service time distribution. This would be acceptable if a single traffic type were to be matched to each medium, but it will not do if we are to accurately reflect the transmission of multiple types of traffic on a single medium.

The remaining queueing disciplines allowable for product-form networks permit separate customer classes in the queue to have separate service time distributions. These three queueing disciplines however (i.e., processor sharing, infinite servers, and last-come, first-served) do not intuitively map well into our concept of traffic queueing at a node, and would not provide an applicable model.

The consequence of all this is that the FIFO queue should somehow be used, but should be limited to serving a single traffic type. Fortunately, this is possible to do in a credible way, and this will be the subject of the next section.

3.2 The Multimedia Network Composite Channel

As was mentioned immediately above, we cannot model separate media channels as separate queues within the constraints of the product-form model unless we attribute the same service time distribution to all traffic using that channel. This would seem to preclude the multiplexing of multiple traffic types (each of which may have its own distinct service time distribution) on a single channel. In order to circumvent this problem, we will instead regard a channel as carrying a single traffic type, and we will in effect multiplex the channels for the traffic type.

In order to explain this concept, we must introduce some notation. Given a single traffic type j at a specific node i, that traffic type is to be multiplexed on the various media available for transmission. Define a traffic/medium multiplexing vector as

$$M^{ij} = (m^{ij}_1, m^{ij}_2, \ldots, m^{ij}_T),$$

where

m^{ij}_k = the proportion of medium k to be devoted to traffic type j at node i, and

T = the number of media available at the node.

Traffic type j at node i will be apportioned as shown on the various available media. This means that the proportion m^{ij}_k of medium k is set set aside

exclusively for traffic of type j.

In effect, this defines a composite channel for traffic of type j. The bandwidth of the composite channel is expressible as the sum of the proportions of the bandwidths of the media channels partially supporting the traffic type. To be precise, if medium k has bandwidth B^i_k at the node i, then the total bandwidth allocated to traffic type j is

$$B^{ij} = \sum_{k=1}^{T} m^{ij}_k B^i_k \qquad \text{(Equation 3 - 1)}$$

The advantage of the composite channel concept is that it presents to the traffic type in question a total bandwidth available, as per the multiplexing scheme of the node, and it allows the representation of the separate traffic types as traveling on separate channels. In this way, all traffic entering a composite channel is of the same type, i.e., has a single service time distribution, and so the queueing discipline associated with the composite channel can be taken to be FIFO.

The composite channel must also be considered from the standpoint of error processes acting on traffic. This will be examined in the next section.

3.3 The Error Process for the Composite Channel

The composite channel comprises, for its related traffic type, a collection of fractions of media channels, each of which may have different error properties relative to the traffic type. The composite channel error rate is therefore dependent on the specific proportions of the various media available to the traffic type for which the composite channel is defined.

Before carrying this reasoning to a precise expression, it will be useful to quantify the error process somewhat more than it previously has been. For each medium/modulation combination, there is some form of signal degradation representing the usual operational characteristics of the medium so modulated. Whatever form this degradation takes, it will affect any specific traffic type to an extent depending on the error-correcting mechanisms built into that traffic type. For the purposes of the current model, we will assume that all traffic types can be thought of loosely as "messages" (the term "packet" seems too dangerously specific), and for each medium/traffic type combination, a missed message rate (MMR) will be determined based on a careful analysis of both the medium/modulation and the traffic type.

Thus for traffic type j and medium k, we will denote a missed message rate (MMR) by E_{jk}. The composite channel can be assumed to carry traffic in direct proportion to the band width allotted per medium type, so the composite error rate for traffic type j at node i should be expressed as

$$E_j = \sum_{k=1}^{T} m^{ij}_k E_{jk} \qquad \text{(Equation 3 - 2).}$$

This error rate is thus interpreted to be an overall missed message rate for the

composite channel. The average missed message rate for all traffic of this type traveling on the composite channel will correspond to this MMR.

3.4 Accounting for Error Correction Traffic

The subsection above dealt with determining the error rate for a composite channel relative to the traffic type that will flow on that channel. The error rate will be applied to determine how many messages (in the present circumstances, the term "messages" will be regarded as a generic term for separate traffic entities) transmitted on the composite channel will be received in unacceptable condition. When received traffic is unacceptable or unusable, there are generally three possible responses to the situation:

(1) the traffic is discarded, with no need for a repeated transmission,

(2) the traffic has substantial forward error correction built in, so the receiving node can resolve the problem with no further use of communications links

(3) the traffic must be retransmitted.

For the purposes of the present model, we are only concerned with processes that increase the burden of the available media. Therefore, we need only concern ourselves with error handling of the third kind. For such error handling processes, we shall assume that each message subject to error correction, when received correctly, is acknowledged by the receiving station. This acknowledgement will generally consist of a short message returned to the transmitting node using the same composite channel. If the received message is not correct, then no acknowledgement is sent.

There are several ways in which the mechanisms of such error handling could be modeled. In keeping with the philosophy of steady-state modeling, we must bear in mind that the purpose of this model is not to follow specific traffic entities through the system, but rather to guage overall traffic congestion and delay through the system. (Actually, since routing in the product-form model is not deterministic, there is no way to account on a message-by-message basis for erroneous traffic transmission.) Therefore, so long as the additional traffic load imposed by error correction is modeled, it is not necessary to actually implement flow paths representing the handling of acknowledgements and retransmissions.

What additional traffic loads are imposed by this error correction scheme? First, there is the additional load arising from the need to transmit repeats of erroneous traffic along the original path. Second, there is the need at the receiving end to generate and return acknowledgements to the transmitting station for correctly received traffic entities. We will account for all effects of erroneous messages by adding an extra load at the transmit node which is equivalent to the additional traffic it must transmit due to the error process. Since the extra load occupies the same FIFO queue as all normal traffic for the affected composite channel, the overall effect on the system is an additional amount of delay in the node due to the need to requeue and retransmit some fraction of the channel traffic. For the acknowledgement process, the extra load is imposed on

the receiving node, since it must use some of its transmit capacity to queue and transmit an acknowledgement.

To adequately account for these added traffic loads, it is necessary to analyze the intensity of traffic flow for the traffic type in question, and then use that intensity figure to calculate the extra traffic loading imposed on the queues in both the transmitting and receiving node. We will do this in the two subsections below.

3.4.1 Erroneous Traffic Effects at the Transmitting Node

If we are dealing with a specific traffic type t, the added load at the transmitting node is a function of its mean length l_t and the missed message rate E_t for the traffic type. In particular, for every original message transmitted, the total load at the transmit node is just

$$S'_t = l_t(1 - E_t) + 2l_t(1 - E_t)E_t + 3l_t(1 - E_t)E_t^2 + \ldots$$

This infinite summation does have a closed form for $0 < E_t < 1$, and yields

$$S'_t = l_t (1 - E_t)^{-1} \qquad \text{(Equation 3 - 3)}$$

In effect, this is the expected length of all traffic generated at this node associated with the original message. By lengthening the nominal traffic length l_t to this value, we have imposed the desired additional load on the node.

3.4.2 Erroneous Traffic Effects at the Receiving Node

We will handle the effects of acknowledgements on the traffic process by also increasing the length of each type t message handled to account for the acknowledgement it requires back to the previous node. However, not all traffic of type t which is in queue at the receiving node has actually been received from other nodes. I.e., the traffic in queue at the node is generally a mixture of received traffic, and traffic originated at the receiving node. Obviously, the node need not generate acknowledgements for traffic internally originated, therefore only some fraction of the type t traffic actually imposes a load on the node. Thus, in order to assess the load at the node correctly, we wish to determine the fraction of traffic of type t originated in the node, relative to all type t traffic processed by the node.

This is not actually a difficult thing to do, since we have available the relative throughputs from Equation 2 - 3 . In terms of the notation of Equation 2 - 3, suppose that P[0,d] represents the originations for traffic type t, and that y_d represents the relative throughput of traffic type t at the receiving node. Then the fraction of traffic which represents local originations of traffic type t, compared to all traffic of type t processed by the node is

$$v_t = P[0, d]/y_d \qquad \text{(Equation 3 - 4).}$$

Then the average length for all type t messages processed by the node is given

by

$$S_t = v_t [l_t (1 - E_t)^{-1}] + (1 - v_t)[l_t (1 - E_t)^{-1} + a_t] \qquad \text{(Equation 3 - 5)}$$

where

a_t = length of the required acknowledgement for traffic type t..

Thus, the overall additional load imposed by the error process is visited on the system effectively by increasing the length of each message to account for its retransmissions by the system, and the acknowledgements sent on its behalf. Thus given the quantities l_t , E_t , and a_t , we can calculate for each node (note that it is a function of each individual node's traffic type t throughput and origination rate) the length S_t for the traffic of type t at that node. This effectively determines the service rate for that traffic type at that node.

Suppose that we are dealing instead with the situation where acknowledgements are sent "out-of-band", i.e., on a channel other than the one which carries the original traffic. Then the additional load on the original traffic channel reverts back to the value S'_t. The remaining part of the traffic generated is the out-of-band component associated with the acknowledgement process, i.e., just the load associated with the generation and transmission of the correct proportion of acknowledgements for the traffic type. That will be given by the difference

$$S_t - S'_t .$$

3.5 Computing Transmission Delays through the Network

The final remaining topic which is to be considered in this section has to do with the means by which path delay through the network can be computed. For the open network product-form model, the computation of all nodal performance metrics is very straightforward once the linear equations (Equations 2 - 3) determining the relative throughputs have been solved. One of the nodal performance metrics available is the mean nodal response time (i.e., queueing delay plus service delay) for each traffic type passing through the node. (See Appendix B for the derivations of system peformance parameters from the relative throughputs.) In order to compute the expected delay for any traffic type along a path through the network, one can add the mean nodal response times for the nodes along that path.

However, if what is desired is an average delay for traffic over a multiplicity of paths, then one cannot simply take the unweighted average of the path delays described in the above paragraph. That is because one cannot assume that equal amounts of the traffic of interest flowed via the various paths. Thus, we must find a means to account for the relative proportion of traffic that flowed along any one path among a collection of paths of interest.

This can be done by reference back to the relative throughputs defined by Equation 2 - 3 . Specifically, let

$$P = < s_1, s_2, \ldots, s_{n(p)} >$$

represent a path through the network, with s_1 being the first node, s_2 being the second node, etc. The expected path delay along this path for traffic type t will be the sum of the expected response times for traffic type t at each node included in the path. Let

$D_t(P)$ = expected delay for traffic of type t traversing path P.

Then if we have another path Q connecting the same origin and destination, a constant α must be such that the expected delay for all type t traffic traveling between these endpoints on both paths can be expressed as

$$E[D_t(P \text{ or } Q)] = \alpha\, D_t(P) + (1 - \alpha) D_t(Q).$$

We determine the constant α from the relative throughputs of the nodes (for traffic type t) and the routing transition probabilities. Beginning at the next to the last node on path P, the proportion of all type t traffic through the node destined for node $s_{n(p)}$ is given by the routing transition probability $P_t[s_{n(p)} - 1, s_{n(p)}]$; multiplying this by the relative throughput $y_{t,n(p)} - 1$ of node $s_{n(p)} - 1$, i.e.,

$$P[s_{n(p)} - 1, s_{n(p)}]\, y_{t,n(p)} - 1,$$

gives a relative measure of the type t traffic traveling this link. Now, from this quantity of traffic, we wish to know what portion arrived at node $s_{n(p)} - 1$ from the preceding node, $s_{n(p)} - 2$. Applying the same reasoning to this situation, we determine a measure of the relative traffic of type t at node $s_{n(p)} - 1$ from node $s_{n(p)} - 2$ as being

$$P_t[s_{n(p)} - 2, s_{n(p)} - 1]\, y_{t,n(p)} - 2\ ,$$

where $y_{t,n(p)} - 2$ is the relative throughput of traffic type t at node $s_{n(p)} - 2$.

This reasoning can be extended inductively backward to the first node of the path, with a similarly defined factor applying at each node. Thus a measure of the relative amount of traffic flowing from node s_1 to node $s_{n(p)}$ is given by the product

$$W_t(P) = \prod_{i=1}^{n(p)-1} P_t[s_i, s_{i+1}]\, y_{t,i}\ . \qquad \text{(Equation 3 - 5)}$$

Thus if the expected delay for type t traffic is to be computed for travel along any of the multiple paths, say $P_1, \ldots, P_n$ it will take the form

$$E[D_t(P_1 \text{ or } \ldots \text{ or } P_n)] = \Big(\sum_{i=1}^{n} W_t(P_i) E[D_t(P_i)]\Big) \Big/ \Big(\sum_{i=1}^{n} W_t(P_i)\Big) \qquad \text{(Equation 3 - 6).}$$

This effectively completes the discussion of model development which was the main subject of this section. All of the technical results presented above are specific to this application, and are generally not part of the generic results derived in product-form network-of-queues expositions. Appendix B provides a full accounting of the generic mathematical treatment of the open product-form network which is sufficient for our purposes.

4.0 INSTRUCTION MANUAL FOR THE MMDESIGN PROGRAM

The main goal of this year's study effort was to develop the mathematics needed to create a credible model of the multimedia network within the product-form network-of-queues framework, and to then implement the concepts in a computer program. The MMDESIGN program developed for this purpose is effectively still a prototype, and will undergo considerable generalization and improvement in the next year. Therefore, the content of this section should not be taken as a permanent record of the capabilities, form, or user-interface associated with this program. Many of the features described in this section are still in development, and others are not yet fully debugged.

The specific objective of this program is to provide an analytical tool enabling communications network designers to assess the tradeoffs involved in assigning various traffic types to various media supporting a multimedia network structure. The tradeoffs relate to the greater or lesser ability of a given medium to service any particular traffic type within the constraints of traffic degradation and delay. Where some media are superior to others relative to these properties, some portion of the traffic may need be relagated to the poorer media. This program will aid analysts in determining the steady-state effects of traffic multiplexing on the media.

MMDESIGN in its present form does not perform any automated optimization of routing. The user can enter the information defining the network, and can then derive and examine the steady-state performance of the system. The primary metrics provided are the expected response times (i.e., queueing delay plus service time) for each node and traffic type, the expected delay times for any traffic type traveling any specific path or collection of paths all of which have the same origin and destination.

This program is quite data intensive, since it will require enough information to completely specify all routing in the network, all traffic types (each of which has its own routing structure), and all media. Thus this program is not "user friendly", in the sense that one can get practical results from its use in short order. To fully specify a large network to the level required for this program could require substantial tedious data input. However, once that data has been supplied, it is possible to examine many traffic multiplexing scenarios with much less expenditure of time and much greater confidence in the results than would be available through simulation.

The following subsection will be devoted to explaining the meaning of each data input required to fully define a multimedia communications network to the program.

4.1 Explanation of Program Inputs

The modeling techniques described in Section 3 permit the network analysis done by MMDESIGN to be done individually by traffic type. That is because the channel multiplexing technique used to create a composite channel makes the

traffic types independent of eachother, except that each traffic type has a limited amount of the total media bandwidth available to it, associated with which is a composite traffic service rate and traffic error rate.

Because this is the case, the data entry process for MMDESIGN is organized primarily around traffic types and the specific information associated with a single traffic type. Furthermore, the information input scheme is such that analysis can proceed for a single traffic type once all of the data associated with that traffic type has been entered.

Since the amount of data to be input for an entire network can be very extensive, the program organization only keeps data for a single traffic type in computer memory at any time. This is of great advantage in the present IBM PC implementation, since most IBM PC's or equivalents have less than 1 megabyte of memory capacity.

Data input for any network design analysis is automatically stored to a file as it is entered. This file can then be invoked in a later session and used as is for further analysis, or edited if it is desired to try a different, but similar network configuration. There is one limitation built into the data storage retrieval process which was unavoidable, given the constraint on available memory, and that is that, although almost any of the originally entered data associated with a network can be edited, the overall "size" of the network must remain the same. In this case, the size of the network is a function of the number of nodes, the number of traffic types, and the number of media types input in the network definition. Once these three values are selected, a new network obtained by editing the present network cannot change any of them. (A larger network can be defined only by going through the full network creation process again.) Thus, if one defines a network, and anticipates that the network later may involve more traffic types, media, or nodes, one should select the maximum values expected for those numbers at the initial creation of the network. Doing so does not measurably add to the workload associated with data entry until such time as the network definition is actually expanded.

The inputs to the program during network creation will now be discussed, in the order in which they are input. First are the <u>global</u> parameters, so called because they are not associated with any single traffic type; these are

a. the total number of communications nodes in the network,

b. the total number of media types in the network,

c. the total bandwidth (in Kbits/second) for each medium type,

d. the total number of traffic types in the network.

Following the entries above, the block of data entries discussed below are all associated with a specific traffic type. The user enters these parameters in consecutive blocks of entries, with all entries in a given block being associated with a single traffic type. After the data for any traffic type has been entered, the user may exit the network creation process and proceed to the analysis portion of the program for the traffic types already defined. The traffic type data entry

comprises

a. the network-wide traffic arrival rate for the traffic type,

b. the mean message length for the traffic type,

c. the length of the acknowledgement for the traffic type (enter 0 if no acknowledgement is used)

d. a collection of missed message rates for this traffic type, one for each medium on which it will be transmitted,

e. a collection of local arrival rates, one for each communications node in the system (these rates correspond to the probability distribution by which the global arrivals for a traffic type are subdivided, as explained in Section 2.2),

f. station traffic multiplexing vectors by which the composite channel for the traffic type are defined (see Section 3.2), one multiplex vector being required for each node in the network,

g. the network routing transition probability matrix P[i, j] for the traffic type, i.e., the defined routing in the system may be varied by traffic type.

The quantity of data required to define a large network is extensive, especially since the items e. through g. must be entered for each traffic type, and some of those items (especially the station multiplex vectors and topology routing matrix) may require substantial numbers of individual entries. However, there is available in the program a copy feature that allows the most voluminous data structures, if identical for different input cases, to be copied from previously entered data. E.g., if, for a given traffic type, all station traffic multiplexing vectors are to be identical, then the copy function allows the analyst to evade a very substantial amount of data entry.

A final data entry process, which is decoupled from network creation/editing, is associated with the determination of the paths over which the model is to compute traffic delay. The inputs in this case specify to the program which network paths are of interest to the user in computing path delay through the network. The user may in effect enter sets of paths, and the final performance output for the program will compute the expected delay for each traffic type along those paths, with the delays computed being averages taken over each path set. The path data need not be entered at the time that the network data described above is entered. The user can enter network definition data, and compute all nodal metrics of interest, if so desired, before proceeding to evaluation of path delays. All path data is entered under a separate menu function "Paths", at a time of the user's choosing. The path data entered is

h. number of sets of paths to be defined,

i. a path description, entered as a sequence of nodes (interpreted as from origin to destination), and the path set in which it is to be included.

As is the case for all network definition data, the path data is stored to disk, and can be recalled and edited at will in association with the network data defining the network.

A final point concerning the total aggregate of input data is that it is possible to enter data in so complex a model as this which is mathematically inconsistent. There are four possible forms of inconsistency , namely,

1. the possibility that the row sums of any routing transition probability matrix do not equal 1 (the row sums are probability densities, and so must add to 1), associated with data entered under item g. above,

2. the possibility that the local arrival rates for a traffic type do not sum to one, associated with data entered under item e. above,

3. the possibility that more than all channel bandwidth for a given media type might be allocated to the various traffic types, associated with data entry under item f. above,.

4. the possibility that a defined path, as entered by the user in connection with item i. above , is not in fact supported by the media routing transition probability matrices for any of the traffic types.

There are "Verify" utilities provided in the program to assist the user in checking that each of the above data types is consistent. A verification function is automatically invoked at the end of a network creation session, to inform the user of any difficulties detected relative to items 1. to 3. above. That utility can also be invoked by the user after any network editing, in order to insure that previously consistent data has not been made inconsistent by the editing process.

Of course, there are other possibilities for what amounts to inconsistent data entry, such as entering parameters which are obviously out of range, or which create hopelessly large traffic loads in the network. There is no range checking for such data errors in the program.

4.2 Program Organization and Menus

This subsection will describe the user interface to the MMDESIGN program, and will explain each program function in detail. It is important to reiterate at this point that MMDESIGN is a prototype program, and will evolve substantially over the next year of this study. Therefore, the material in this manual concerning program interface and function is interim information.

4.2.1 User Interface Format

The MMDESIGN program is entirely menu-oriented. It consists of a main menu which requires single keystroke responses from the user, and several submenus associated with main menu commands. The general format of all menu lines is the same: each command on a menu line is written in the form

"X(xxxxxx ",

and the user must enter the first letter of the command (in either upper or lower case) , followed by the "ENTER" key. This results either in the presentation of a submenu or specific prompts for data entry associated with the command. In general, all possible user actions are met with clear prompts for the appropriate action.

The other major aspect of MMDESIGN screen format is the division of the screen into two portions. The top portion comprises about one-fifth of the total screen, and provides a status/navigation window to the user. This window displays at any time the current menu level at which the user is active (shown hierarchically from the top menu), as well as the name of the network file and the traffic type currently under investigation. If no network file has been opened, then the name displayed is "Undefined".

The lower portion of the screen is the user/program interaction screen, and effectively functions as an ordinary terminal interface, with data scrolling off the top when the screen becomes full, and new data is entered at the bottom.

The program contains, together with the verify functions, a number of other warnings indicating fatal problems, such as an inability to open a requested file. Warnings of this type are presented in blinking red text, and are preceded by a short tone from the computer speaker.

4.2.2 Interpretation of the Menus

In this section, each of the MMDESIGN commands available through the program menus will be explained. Since the menu structure is hierarchical, menus at lower levels may be referred to with simultaneous reference to their ancestors in the hierarchy, where that improves the clarity of the presentation. Such references will take the form "PARENT/CHILD/GRANDCHILD/...". In this notation, the main program menu will be referred to as "MAIN". While reading to the material below, the reader may find it helpful to refer to the template in Appendix A which provides a view of the full hierarchical menu structure of the MMDESIGN program.

The MAIN menu is presented as the following line:

" N(ew, C(reate, E(dit, P(rint, R(ecall, T(hruputs, P(aths, M(etrics, Q(uit: "

To facilitate document organization, the discussion will be broken into separate subsections below. It should be noted that a thorough reading of the following subsections is mandatory before attempting use of MMDESIGN, because many essential details of program operation are embedded in the following discussion, and may effect the understanding of commands other than those under which they are introduced.

4.2.2.1 The "N(ew" Command

In order to perform any actions on a network, the user must supply a network name to the system. This name is the same as the filename in which the network data is to be stored, but the user does not supply the extension to the filename. In effect, the network will create three files files with the same root name, but different extensions. These three files will be

1. "NetworkName.top", which contains the topology and network definition data associated with data entry items a. - g. discussed in Section 4.1.

2. "NetworkName.thp", which contains the relative throughput data for all traffic types defined by the user,

3. "NetworkName.pth", which contains the path sets associated with data entry items h. and i. discussed in Section 4.1.

These three files will be stored in whatever the current DOS directory was at the time of program initiation. When there is a need for the program to open these files, the program looks only in the current directory , i.e., the directory that was active at program initiation, or any other directories made available by the DOS "PATH" command.

In general, the program will prompt the user for a network file name if one has not been defined and the current requested action requires one. Once that name has been supplied to the program, it remains the current network file for all further actions unless the "N(ew" command is invoked. The "N(ew" command is the means by which the user can change from one network file to an unrelated one, if that is desired. Note that invoking the new command does not actually store or retrieve any data to memory, but only establishes that all further actions will refer to a different network file. The means by which data storage is accomplished in the program prevents any loss of data in any event: all relevant data for a network analysis is always stored as soon as it is created, so that errors on the part of the user concerning possible loss of data are minimized.

4.2.2.2 The "C(reate" Command

The "C(reate" command is used when a new network, not previously defined and stored to disk, is to be created. If no network name has yet been defined via the "N(ew" command, the user will be prompted to supply a network name. If a previous file of the same name already exists in the active directory on disk, then the user will be warned, and given the option to discontinue. (Continuing at this point will erase the previous file of the same name already on disk.) Once the filename has been selected, the "C(reate" function steps the user through all data entry associated withthe full definition of a network structure, with data entry being required for items a. - g. in Section 4.1, and the order of entry being in the order indicated.

The data needed to define a large network can be quite voluminous, so MMDESIGN provides certain shortcuts to the user to eliminate the entry of redundant or assumed values. This applies specifically to the following types of data.

1. Entry of error rates for all media and a specific traffic type can be eliminated if all such entries are identical with those for a previously defined traffic type. MMDESIGN asks if the current entries are like those for a previous traffic type, and if so, allows the user to input the traffic type index only. Then the previous error rate data is copied to the current traffic type.

2. Local arrival rates for the traffic type, i.e., the specific probabilities that a newly originated message will be associated with a given node, can also be copied from one traffic type to a later traffic type, by the mechanism described above.

3. The multiplex vectors, by which the composite channel for a given traffic type is defined (data item f. of Section 4.1) can be copied from one traffic type to another by the mechanism described above.

4. The topology of the network also is unique to each traffic type (i.e., each traffic type may adhere to a separate matrix of routing transition probabilities), but the routing matrix of a previous traffic type can be copied to the current type by the mechanism described above.

A final data entry economy is associated with the entry of specific routing probabilities for the routing transition probability matrix associated with a traffic type: namely, all entries of the probability matrix are initialized to zero, so the user need only enter the data associated with actual links in the network. For those data entries required, data is entered on a single line, as the origin node, destination node, and probability, in that order, separated by spaces and terminated by a carriage return.

The "C(reate" command steps the user through the input of all required data, looping through the traffic type specific data until all traffic types have been defined. When the data entry process is complete, it automatically verifies the consistency of the data, and provides a screen warning if any inconsistencies are found. This screen warning does not pinpoint the nature of the data inconsistency, however: the user should invoke the "MAIN/Edit/Verify" command in order to get a detailed account of where the inconsistencies were found.

A final important point is that the user need take no action to insure that a created network definition is stored to disk. The storage process is carried out simultaneously with data entry, and is automatically completed and the file closed at the termination of network creation.

4.2.2.3 The "E(dit" Command

Invoking "E(dit" at the MAIN menu level confronts the user with a new menu,

"E(dit, V(erify Q(uit:: "

The "MAIN/Edit/E(dit" command is used to modify a previous network definition stored in a network file. The file to be edited will be the current one, as shown in the Navigation/Status window, or, if none has been identified, the

"MAIN/Edit/E(dit" command will prompt for a file name. All of the network parameters in a file may be modified, with the exception of the number of network nodes, the number of traffic types, and the number of media types. (A later version of MMDESIGN will permit modification of these parameters also.)

Invoking the "MAIN/Edit/Edit" menu results in yet another menu of the form

" Edit functions are as follows:
0: Exit the edit function,
1: modify media bandwidths,
2: modify traffic type global arrival rates,
3. modify traffic type length,
4. modify traffic type acknowledgement length,
5: modify traffic type/medium type error rates,
6. modify traffic type station arrival rates,
7. modify traffic type medium multiplex vector,
8. modify traffic type station connectivity.

When the user invokes any of these choices except "0", the program prompts for information relating to the specific data type to be modified. Some of these choices, on the assumption that the user will wish to modify a multiplicity of them at one time, result in a menu of their own, which allows sequential modification of the data type in question, or a "Q(uit" option to return to the "MAIN/Edit/Edit" menu.

A final point concerning editing is that the user does not in fact edit the original data file during the actual edit session. Instead, a temporary file of the same root name, but with extension ".tmp" is created, and all editing changes are made to that file. When the user invokes the "MAIN/Edit/Edit/0" command to exit an editing session , the program provides the option of storing the edited data under the original file name, under a new file name, or abandoning the changes with no permanent file being created. If a new file name is chosen, it does not automatically become the active network of the program. It will be necessary for the user to use the "MAIN/New" command to select the new file for analysis.

Invoking the "MAIN/Edit/Verify" command provides the user with the opportunity to check the current network data file (i.e., the one displayed by the program in the Navigation/Status window) for consistency. The verify command provides specific outputs to screen and printer, if requested, indicating the nature and locations of the inconsistencies found. Data which contains inconsistencies will not provide reliable network performance metrics, and in fact may cause system crashes when computation based on it is attempted. The user must edit inconsistent data, and reverify it to insure that the results of analytical endeavors with MMDESIGN are meaningful

4.2.2.4 The "H(ardcopy" Command

The "H(ardcopy" command on the MAIN menu enables the user to get hardcopy output of the network definition data. Invoking the "H(ardcopy" command presents the user with another menu.

"Display G(lobal data, T(raffic type data, A(ll data, Q(uit: ".

The user can print out only that network data which is global (data items a. - d. in Section 4.1), only data that is specific to one traffic type (items e. - g. in Section 4.1), or the entire contents of the active network definition file. The printout is formatted so that the various data types contained in the file are easily distinguishable. If no data file is currently active, the program will prompt for one.

4.2.2.5 The "R(ecall" Command

The "R(ecall" command permits the user to bring into computer memory the global data from a network definition file, and the traffic type data in that file associated with one specific traffic type index. (Because the product-form analysis of even modest-sized networks requires a substantial body of data, the data for only one traffic type at a time is ever in memory. All analysis needed for that traffic type can be done from that data.) Invoking the "R(ecall" command establishes the recalled network and traffic type as the currently active data set in the program. The "R(ecall" command simultaneously brings any throughput data already computed for the traffic type into memory, (see 4.2.2.6 and Equation 2 - 3), so that the user may pursue the analysis of the traffic type.

4.2.2.6 The "T(hruput" Command

The "T(hruput" command (mispelled to save space in the menu line), is used to compute the relative throughputs for the network and traffic type currently active in the program. The relative throughputs for any traffic type are computed from Equation 2 - 3, and, once obtained, are the basis for all performance metrics available for analysis. Invoking the "T(hruputs" command begets the user another menu,

"C(ompute, D(isplay, P(rint, Q(uit:"

These menu entries permit the obvious actions to be performed, where the "P(rint" command may be used to print throughputs for one, or all, traffic types.

Since the throughput computation is the most intensive computation required for open network analysis, the results of a successful throughput computation are automatically stored to a file the name of which has the currently active network name as root, and the extension ".thp". Thus if a user wishes to terminate an analysis session, to be resumed at a later time, it will not be necessary to recompute these quantities, which computation may prove to be time-consuming for large networks.

4.2.2.7 The "P(aths" Command

The paths selected for analysis in the network can be chosen independently of the original network creation, editing, and throughput calculation processes. In the normal order of events, the analyst would define a network via the creation process, edit it if necessary to delete inconsistencies, and then compute the

throughputs associated with that network definition for the traffic types of interest. After those activities had been completed, the analyst could directly examine the metrics associated with the individual nodes of the network (see Section 4.2.2.8 below for a description of the available metrics). However, in order to understand the effect of the network structure on the end-to-end delay of traffic, the analyst must examine the delay times of multiple-node paths. If a particular end-to-end scenario of interest involves only one path, then the end-to-end delay is simply the sum of the nodal response time (i.e., queueing plus service delay) for the path.

However, if the end-to-end scenario involves an origin and destination connected by multiple paths, then the analyst might desire the mean delay for all traffic of a given type between the origin and destination, traveling by whatever path is available. This was discussed in detail in Section 3.5, where it was shown that the computation of expected delay requires in such case a weighted sum of path delays, for all paths regarded as routes between the origin and destination. The "P(aths" command permits the analyst to define sets of paths from a specific origin to a specific destination, such that any such set of paths will be taken as a collection over which expected end-to-end delay is to be computed. These sets can be created, edited, and verified using the "P(aths" command.

When the analyst invokes the "P(aths" command,the menu line

"C(reate, A(dd, D(elete, V(erify, H(ardcopy Q(uit: "

appears. The explanation of these options will be taken up in their order of appearance.

First, the "MAIN/Paths/C(reate command permits the analyst to establish a new set of paths between any origin and destination node, and to list the traffic types of interest for that set of paths. The user is informed (based on how many path sets have already been defined) of what integer index will be associated with this path set, and then is prompted for the number of paths to be included in the set. Following that, the user enters the individual paths, each as a sequence of nodes separated by spaces, all nodes of a given path being entered sequentially from origin to destination, separated by spaces, and terminated with the "ENTER" key.

The user is then prompted for the traffic types of interest relative to this path (i.e., the traffic types for which the expected aggregate end-to-end traffic delay is to be computed), which are to be entered separated by spaces, and terminated with the "ENTER" key. At the end of the path creation process, the program automatically checks that the paths do indeed exist (i.e., all links of each path have an associated nonzero routing transition probability), and informs the analyst of any inconsistencies discovered.

The "MAIN/Paths/A(dd" and "MAIN/Paths/Delete" commands constitute the editing process for the library of stored path sets. The "A(dd" command allows the analyst to insert additional paths in existing path sets. The program prompts for the path set to which a new path is to be added, and, following the response, the analyst enters a path description in the same format as was described for the path set creation process. The additional path is automatically verified as valid

(in the same manner as for path creation), and the user is warned of inconsistency.

For the delete command, the analyst is prompted for a path set from which to delete a path, and then is shown a screen listing of the paths in the set. The user enters a path index for the path, as adduced from the screen listing of the paths.

Note that the path sets defined for a network are stored in a separate file, the root name of which is the network name, and the extension for which is ".pth". All creation and modiffication activities involving the path sets automatically update this file without user intervention.

The "MAIN/Paths/Verify " command can be called at any time by the analyst, and will either verify a specific path set as containing consistent paths, or will verify all existing path sets.

The "MAIN/Paths/Hardcopy" command permits the user to obtain a printer output of the contents of either a specific path set, or all path sets.

The "MAIN/Paths/Quit" command returns the user to the MAIN menu line.

4.2.2.8 The "M(etrics" Command

The "M(etrics" command allows the analyst to examine the various network peformance metrics which can be computed for the network. All of these metrics require first that the relative throughputs associated with Equation 2 - 3 and section 4.4.2.6 have been computed. All metrics related only to nodal performance can then be obtained directly: those relating to mean delay for traffic traveling along paths, or collections of paths, cannot be computed until the desired sets of paths have been defined via the "P(aths" command discussed in Section 4.4.2.7.

Invoking the "M(etrics" command presents the menu

"N(odes, P(aths, Q(ueue Length Density, H(ardcopy, Q(uit: ".

The "H(ardcopy" option does not prompt the user, but simply turns the printer on so that a hardcopy record of all metrics requested is provided. This hardcopy option remains activated until the "MAIN/Metrics/Quit" command is invoked. Hardcopy requested is formatted so that it is clear from the printout exactly what performance metrics have been supplied.

The "N(odes" command presents the user with a menu line

"S(ingle Node, A(ll nodes, Q(uit: ".

The user can exercise the "S(ingle Node" option to request the available performance metrics for a specific node, and may request the "A(ll Nodes" option to get a listing of the nodal performance metrics for all nodes. The screen listing of metrics scrolls, as does all ordinary screen output, so it is advisable to have invoked the "MAIN/Metrics/Hardcopy" option prior to invoking the "All Nodes"

option. In either the "All Nodes" or "Single Node" case, the metrics supplied for each node are

1. the throughput of the node for each traffic type,
2. the total throughput of the node for all traffic types combined,
3. the utilization of each composite channel at the node,
4. the utilization ot the individual media at the node,
5. the nodal response time for each traffic type at the node.

These measures will be given more precise mathematical definitions in Appendix B. Taken together, they provide a good diagnostic tool by which the analyst can examine bottlenecks in the network, and determine their causes.

Invoking the "MAIN/Metrics/Paths" command presents the user with another menu,

"S(ingle Path Set, A(ll Path Sets, Q(uit: "

The "S(ingle Path Set" option prompts the user to enter the identity of a single path set (path sets are discussed in connection with Section 4.2.2.7), and then the overall expected path delay for the aggregate of all paths in the set is computed and output to the screen and/or printer.

The "A(ll Path Sets" option outputs the same metrics as the "S(ingle Path Set" option, but does so for every path set which is in the paths file for the network. The delays are provided for each traffic type which was associated with the path set of interest at the time the path set was defined.

The numbers supplied in this case are just the mean delay time for transit of the traffic type(s) from the origin to the destination node. In a later version of the program, the computation of variance for that delay will also be supplied.

The "Q(ueue Length Density" command provides a more resolved look at the potential queueing bottlenecks in the system. When this command is invoked, the analyst is prompted for a node number and traffic type, and the program then computes and outputs the probability density of the queue length for that traffic type at that node. In theory, this density has infinitely many non-zero terms, but in practice, the terms are truncated when the queue length probabilities become less than 10^{-6}.

Invoking the "MAIN/Metrics/Quit" command returns the analyst to the MAIN program menu.

4.2.2.9 The "Q(uit" Command

Invoking the "Q(uit" command at the MAIN menu level exits the main program. Since all creation and editing processes in MMDESIGN are stored to files as they occur, the user may exit the program without first being concerned about data changes which may have been made during the program.

4.3 Summary and Further Directions

This completes the discussion of the program menus, and should provide the analyst with enough background to successfully exploit the power of MMDESIGN to examine the overall traffic flow in a network, and to seek better allocation of assets. The MMDESIGN program must be developed and used in prototype fashion over some range of test cases in order to fully understand its potential as an adjunct to network design by simulation. The second year of this study will be focused on studying such test cases with MMDESIGN, and, using the insight thus gained, creating automated capabilities within MMDESIGN to seek allocation of assets so as to achieve optimum traffic timeliness within the bandwidth constraints of the system.

5.0 REFERENCES

[1]. Aigner, Martin, *Combinatorial Search*, John Wiley & Sons, New York, NY, 1988

[2]. Nemhauser, George L., and Wolsey, Laurence A., *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, NY, 1988

[3]. Jackson, R. R. P., "Queueing Systems with Phase Type Service", J. R. Statistical Society, B18, 129-132

[4]. Baskett, F., Chandy, K., Muntz, R., Palacios, F., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", J. ACM, Vol. 22, No. 2, Apr '75, pp. 248 - 260

[5]. Lemoine, A. J., "Networks of Queues, a Survey of Equilibrium Analysis", Management Science, Vol. 24, No. 4, Dec. '77, pp. 464 - 481

[6]. Lavenberg, S.S. (Editor), *Computer Performance Modeling Handbook*, Academic Press, San Diego, CA, 1983

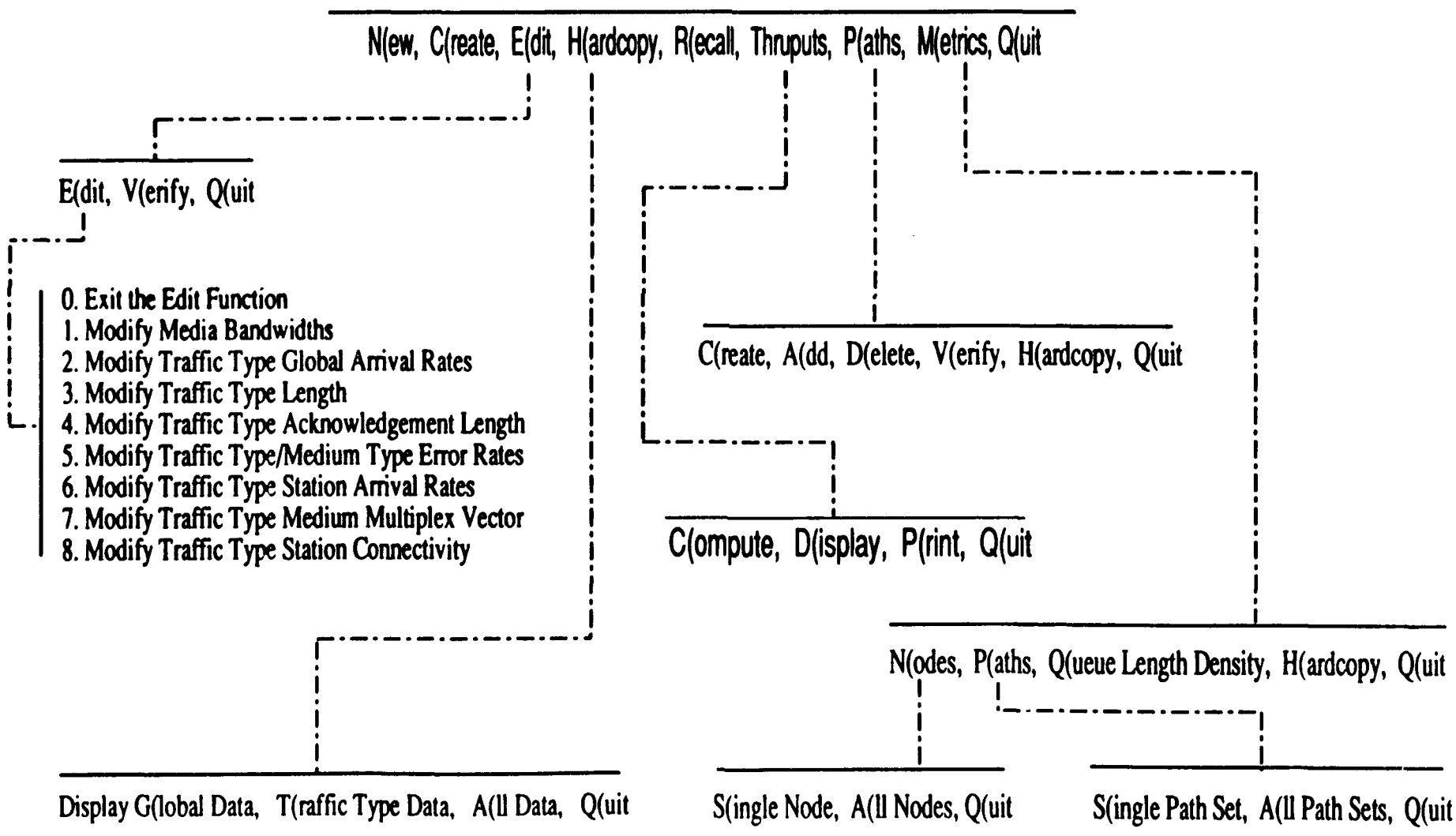
[7]. Akyildiz, I. F., "Exact Product-Form Solution for Queueing Networks with Blocking", IEEE Trans. on Computers, Vol. C-36, No. 1, Jan '87, pp 122-125

[8]. Conway, E. A., Georganas, N. D., "RECAL - A New, Efficient Algorithm for the Exact Analysis of Multiple Closed-Chain Queueing Networks", J. ACM, Vol. 33, No. 4, Oct. '86, pp. 768 - 791

[9]. DeSilva, E. D., Lavenberg, S. S., "Calculating Joint Queue -Length Distributions in Product-Form Queueing Networks", J. ACM, Vol. 36, No. 1, Jan '89, pp. 194 - 207

APPENDIX A

VIEW OF THE HIERARCHICAL MMDESIGN MENU STRUCTURE



APPENDIX B

THEORETICAL DESCRIPTION OF CLOSED-FORM MODELING FOR OPEN NETWORKS OF QUEUES

The general description of the context and limitations of closed-form network-of-queues models were discussed in Section 2. The techniques by which such modeling can be fruitfully applied to the design of multimedia communications networks were discussed in Section 3. In that section, an open network model was adopted for the study of the multimedia traffic type issues of communications. It is fortunate that the open network model is the germane model in this case, since the computational difficulties associated with that model are less severe than for closed networks. (Recall that an open network allows arrivals to and departures from the system, while a closed network does not: thus the customer count for a closed network does not change.)

This appendix will provide careful mathematical development of the essential expressions for the primary performance measures of open network models. The development presented in this appendix is the essential underlying mathematics on which the MMDESIGN program is based.

To begin this exposition, recall that the assumption underlying the success of the closed-form technique is that the network have a product-form. This assumption actually means that

1. the state of each node is expressible solely in terms of its current customer population, i.e., in terms of the numbers of customers of each type currently in queue and in service,

2. the state of the entire network is expressible exactly in terms of the individual states of the nodes.

The latter assumption is expressible in terms of a product of independent probabilities,

$$Pr\{(S_1, S_2, \ldots, S_N)\} = Pr\{S_1\}Pr\{S_2\}\ldots Pr\{S_N\}$$

where

S_i is a vector representing the customer population at node i,

$Pr\{(S_1, S_2, \ldots, S_N)\}$ is the probability of the network having the aggregate state represented by the state vectors of the nodes, and

$Pr\{S_i\}$ is the probability that node i has the state represented by S_i.

These assumptions hold true provided that the routing in the network is probabilistic rather than deterministic, i.e., that each customer leaving a node has a probability of thereafter going to any other connected node. The routing probabilities are grouped in a routing transition probability matrix, which may take the form

P[(i,s), (j,t)] = probability of a customer of class s at node i transits to a customer of class t at node j .

However, the pairwise notation used above is inconvenient, so we opt instead to use a notation where each (node, customer class) pair takes on a single index notation, which we will call the customer type. In effect, each customer class has now been subdivided into many customer types. (There are then many more customer types than classes, but the matrix dimensions remain the same.)

Given this change of notation, the matric expression for routing transition probabilities becomes

P[c, d] = probability that a type c customer transits to a type d customer.

This matrix would be a square matrix, with dimension equal to the total number of customer types determined in this way. However, for open networks, we include the possibility that a message leaving processing at a node may be absorbed at that node. This effectively adds a "zero-th" customer type, which is included as a zero-th column of the matrix, P[c,0].

Together with this matrix, the arrival rates at the nodes determine the essential loading of the network, and from this, all measures of performance are derived. For reasons of mathematical convenience, the arrival process is expressed in terms of a global arrival rate per customer class, which is the total arrival rate for all customers of that class to all nodes, and a probability distribution which subdivides that arrival rate between all relevant nodes. The global arrival rate is taken to be Poisson (i.e., exponentially distributed interarrival times). Each customer class global arrival rate can in fact be dependent on the number of customers of that class already in the system: thus the arrival rate for customer class i could be expressed as a discrete function

$$\lambda_i(0),\ \lambda_i(1),\ \lambda_i(2),\ \ldots\ .$$

In the application of this theory to multimedia communications, there has been no need to consider variable arrival rates, so we will denote the arrival rate for customer class i simply by λ_i .

Now the global arrivals into customer class i are partitioned by a discrete probability density, say $p_i = (p_{i1}, p_{i2}, \ldots, p_{iN})$, where

p_{ij} = the probability that an arrived class i customer arrives at node j .

The actual consequences of this two-step description of arrival is actually equivalent to postulating Poisson arrivals for each customer class at each node, where the overall arrival rate for customer class i at node j becomes

$$\lambda_{ij} = p_{ij}\lambda_i. \qquad \text{(Equation B - 1)}$$

Converting over to customer types, where d is a customer type which is of customer class i, we will use the notation

P[0, d] = probability that a customer class global arrival of class i goes to customer type d.

In this notation, we are prepared to state what is the fundamental relationship from which all of the remaining performance measures flow, namely,

$$y_d = P[0,d] + \sum_{c \in C} y_c P[c,d] \qquad \text{(Equation B - 2)},$$

which expresses the relationship of the relative throughputs for the network. In this equation,

y_d = the relative throughput of customer type d, and
C = the set of all customer types (including the "0" type).

The equations represented by Equation B - 2 are a set of linear equations which, for open networks (i.e., at least one P[0, d] not equal to 0), are uniquely solvable for the quantities y_d, $d \in C$. In many instances, the equations in fact decompose into disjoint subsets of equations, because the potential classes and nodes that some subsets of customers might visit are restricted by routing limitations to less than the full sets of nodes and classes. Such subsets are called closed routing chains.

The quantities y_d are called throughputs because Equations B - 2 are effectively flow equations, and the y_d correspond to the total traffic intensity entering a node from all other sources. These throughputs are called "relative" because the equations do not involve in any way the global arrival rates to the system: however, the only effect of the global arrival rates is to scale the absolute traffic throughput values to some factor times the relative throughputs. There may be several of these customer "type" throughputs associated with a node, of course.

Once Equations B - 2 have been solved for the y_d, several derived performance measures for the nodes are available, as described below. For node j and customer type c, let

C_j = the set of all customer types passing through node j, and

$E[S_c]$ = the expected service demand of a customer of type c.

Then

$$y_{(j)} = \sum_{c \in C_j} y_c \qquad \text{(Equation B - 3)},$$

represents the total relative throughput of node j.

$$E[S_{(j)}] = \left[\sum_{c \in C_j} y_c \, E[S_c] \right] / y_{(j)} \qquad \text{(Equation B - 4)},$$

represents the expected service demand per customer on node j independent of customer type.

$$b_c = y_c \, E[S_c] \qquad \text{(Equation B - 5)},$$

represents the total customer type c service demand on node j.

$$b(j) = \sum_{c \in C_j} b_c \qquad \text{(Equation B - 6),}$$

represents the total expected service demand for node j.

Again, all of these numbers are relative quantities: they provide comparisons between nodes, but until they are multiplied by the absolute arrival rates, they do not provide absolute values for the indicated quantities. If λ_i represents the absolute arrival rate for type c customers (which are of class i), then we can express absolute arrival rate for type c customers as

$$\Lambda_c = \lambda_i \gamma_c \qquad \text{(Equation B - 7),}$$

and we can express the type c absolute service demand as

$$\rho_c = \lambda_i b_c, \qquad \text{(Equation B - 8)}$$

and the absolute service demand on node j is

$$\rho'_{(j)} = \lambda_i b_{(j)} \qquad \text{(Equation B - 9).}$$

The above expressions do not reflect the fact that the service rates at the nodes can also be taken to be dependent on the number of customers already in queue at those nodes. However, we have no need of queue-dependent service rates in the MMDESIGN program, so we shall not consider the extra mathematics associated with that case.

We are now in a position to express the probability density for the queue length at a node. If μ_j is the service rate at node j, and n_j is the number of customers currently at node j, then

$$\text{Prob}\{n_j = q\} = [\, \rho'_{(j)} / \mu_j \,]^q . \text{Prob}\{n_j = 0\} \qquad \text{(Equation B - 10).}$$

For the FIFO queue, which is the only queue of interest in the MMDESIGN program, the latter factor is given by

$$\text{Prob}\{ n_j = 0 \} = 1 - (\lambda_i b_{(j)} / \mu_j) \qquad \text{(Equation B - 11).}$$

The latter term in the last equation is usually called the traffic intensity at a node, and we will denote it as

$$\rho_j = \lambda_i b_{(j)} / \mu_j \qquad \text{(Equation B - 12).}$$

From the above results for FIFO queues, it is possible to express the expected queue length in closed form, i.e.,

$$E[n_j] = \rho_j / (1 - \rho_j) \qquad \text{(Equation B - 13).}$$

The above results essentially provide the mathematical elements underlying the derivation of performance metrics for the MMDESIGN program.

APPENDIX C

MMDESIGN SOURCE CODE

The MMDesign source code is written in Borland's Turbo Pascal. The source code consists of a main program and two supporting units, as follows.

MMDesign.PAS -- the source code for the main program,

Data_IO.PAS -- the source code which supplies all the file handling and data manipulation activities for the main program

NetComp.PAS -- the source code which supplies the computational fucntions needed by the main program.

Note that MMDesign is an evolving program, and thus the source code supplied in this appendix will undoubtably be considerably expanded and altered following publication of this document.

```
PROGRAM MMDesign;              (*John R. Doner  8 August 1989*)

(*This program is the main implementation of the networks of queues theory
  as applied to the Multimedia Network Design Study.  Note that this code
  is in an evolutionary state, and as such includes partially implemented
  and unimplemented features.*)

USES Data_IO, NetComp, CRT;

(*************************************************************************
                DICTIONARY OF SIGNIFICANT PROGRAM VARIABLES

  BadFile       -- controls exit from a procedure if data file not available
  command        -- user input to any menu prompt
  IOWindow       -- denotes the main window on the screen for user input/output
  message        -- used to pass string to CenterText procedure
  NetDefined     -- specifies whether a network is currently in memory
  NetworkName    -- name of currently active network
  NoGo           -- general purpose flag, used variously in program
  Print          -- controls hardcopy output from the Data_IO.Verify procedure
  quit           -- controls exit from the main menu program loop
  TrafficIndex   -- denotes traffic type currently of interest

*************************************************************************)

VAR NetworkName, message                                  : STRING;
    TrafficIndex, i                                     : INTEGER;
    quit, BadFile, Print, NoGo, NetDefined                : BOOLEAN;
    command                                              : CHAR;
    IOWindow                                             : TEXT;

(*The following procedure provides the top-of-screen display on the screen,
  indicating the current status of the program.*)

PROCEDURE NewScreen(title: STRING);
VAR spaces: INTEGER;
    Xtop, YTop, XBottom, YBottom, BackColor, ForeColor, StatusColor: BYTE;
    Stat: TEXT;

BEGIN

  (*Open the status window and write display to it *)

  XTop := BYTE(1);
  YTop := BYTE(1);
  XBottom := BYTE(80);
  YBottom := BYTE(7);
  BackColor := BYTE(13);
  ForeColor := BYTE(14);
  StatusColor := BYTE(15);
  TextBackground(BackColor);
  TextColor(ForeColor);
  Window(Xtop, YTop, XBottom, YBottom);
  AssignCRT(Stat);
  REWRITE(Stat);
  ClrScr;
  Write(Stat,'A +++++++++++++++  AIRMICS MULTIMEDIA NETWORK DESIGN PROGRAM  ');
  WriteLn(Stat,'+++++++++++++++ A');
```

```
  WriteLn(Stat, 'I',' ':77, 'I');
  spaces := (40 - Length(NetworkName)) DIV 2;
  Write(Stat, 'R',' ':spaces, 'Current Data File: ');
  TextColor(StatusColor);
  Write(Stat, NetworkName);
  TextColor(ForeColor);
  Write(Stat, '  Traffic type:');
  TextColor(StatusColor);
  Write(Stat,TrafficIndex:3);
  TextColor(ForeColor);
  IF (2*spaces + 39 + Length(NetworkName)) < 79 THEN spaces := spaces + 1;
  WriteLn(Stat, ' ':spaces, 'R');
  WriteLn(Stat, 'M', ' ':77, 'M');
  spaces := (63 - Length(title)) DIV 2;
  Write(Stat, 'I',' ':spaces,'*** Menu: ');
  TextColor(StatusColor);
  Write(Stat, title);
  TextColor(ForeColor);
  IF (2*spaces + 16 + Length(title)) < 79 THEN spaces := spaces + 1;
  WriteLn(Stat,' ***', ' ':spaces, 'I');
  WriteLn(Stat, 'C', ' ':77, 'C');
  Write(Stat, 'S ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++');
  Write(Stat, '+++++++++++++++ S');
  CLOSE(Stat);
END(*NewScreen*);


(*The following procedure opens the main I/O window for data entry.*)

PROCEDURE MainWindow;
VAR  XTop, YTop, XBottom, YBottom, BackColor, ForeColor: BYTE;
BEGIN
  XTop := BYTE(1);
  YTop := BYTE(8);
  XBottom := BYTE(80);
  YBottom := BYTE(25);
  BackColor := BYTE(7);
  ForeColor := BYTE(1);
  TextBackGround(BackColor);
  TextColor(ForeColor);
  Window(XTop,YTop, XBottom, YBottom);
  AssignCRT(IOWindow);
  REWRITE(IOWindow);
  ClrScr;
  WriteLn(IOWindow)
END(*MainWindow*);

BEGIN(*MAIN PROGRAM*)

  (*Initialization of program status parameters*)

  NetworkName := 'Undefined';
  TrafficIndex := 0;
  NetDefined := FALSE;
  quit := FALSE;

  REPEAT
    NewScreen('MAIN');
    MainWindow;
```

```
Write(IOWindow, ' N(ew, C(reate, E(dit, H(ardcopy, R(ecall, T(hruputs,');
Write(IOWindow,' P(aths, M(etrics, Q(uit: ');
RESET(IOWindow);
ReadLn(IOWindow, command);
CASE command OF
 'N','n':
 BEGIN
   NewScreen('MAIN/New');
   MainWindow;
   Write(' Enter new network name: ');
   ReadLn(NetworkName);
   NetDefined := TRUE
 END(*CASE 'N'*);
 'C','c':
  BEGIN
    NoGo := FALSE;
    NewScreen('MAIN/Create Network');
    MainWindow;
    Write(' Enter the filename in which network data is to be stored: ');
    ReadLn(NetworkName);
    NewScreen('MAIN/Create Network');
    MainWindow;
    NetDefined := TRUE;
    CreateNetwork(NetworkName);
    TrafficIndex := NumberTrafficTypes
  END(*CASE Create*);
  'E','e':
  BEGIN
    REPEAT
      NewScreen('MAIN/Edit');
      MainWindow;
      IF NOT NetDefined THEN
      BEGIN
        Write(' Enter name of file containing network data: ');
        ReadLn(NetworkName);
        NetDefined := TRUE;
        NewScreen('MAIN/Edit');
        MainWindow
      END;
      Write(' E(dit,  V(erify,  Q(uit: ');
      ReadLn(command);
      CASE command OF
        'E','e':
        BEGIN
          NewScreen('MAIN/Edit Network Data');
          MainWindow;
          EditNetwork(NetworkName);
        END(*CASE 'E'*);
        'V','v':
        BEGIN
          NewScreen('MAIN/Edit/Verify Network Data');
          MainWindow;
          WriteLn;
          WriteLn('                    **** Network Data Verification ****');
          WriteLn;
          Write(' Is hardcopy output desired? (y/n): ');
          ReadLn(command);
          WriteLn;
          IF command = 'y' THEN Print := TRUE ELSE Print := FALSE;
```

```
        IF NOT Verify(NetworkName, Print) THEN
        BEGIN
          WriteLn;
          BEEP;
          TextAttr := BlinkOn OR TextAttr;
          WriteLn('              **** WARNING: data must be edited before use. ****');
          TextAttr := BlinkOff AND TextAttr;
          WriteLn
        END
        ELSE Write(' Network data passes all consistency tests.');
        Write(' Press any key to continue.');
        ReadLn
      END(*CASE 'V'*);
      'Q', 'q': command := 'q'
    END(*MAIN/Edit CASES*)
  UNTIL command = 'q'
END(*CASE Edit*);
'H','h':
BEGIN
  NewScreen('MAIN/Hardcopy');
  MainWindow;
  IF NOT NetDefined THEN
  BEGIN
    Write(' Enter name of network data file: ');
    ReadLn(NetworkName);
    NetDefined := TRUE
  END;
  WHILE command <> 'q' DO
  BEGIN
    NewScreen('MAIN/Hardcopy');
    MainWindow;
    BadFile := FALSE;
    WriteLn;
    Write(' Display G(lobal data, T(raffic type data, A(ll data, Q(uit: ');
    ReadLn(command);
    CASE command OF
      'G','g':
      BEGIN
        IF NOT DisplayNetwork(0, NetworkName) THEN BadFile := TRUE
      END(*CASE 'g'*);
      'T','t':
      BEGIN
        WriteLn;
        Write(' Enter traffic type for which hardcopy is desired: ');
        ReadLn(i);
        IF NOT DisplayNetWork(i, NetworkName) THEN BadFile := TRUE
      END(*CASE 't'*);
      'A','a':
      BEGIN
        IF NOT DisplayNetwork(0, NetworkName) THEN BadFile := TRUE;
        FOR i := 1 TO NumberTrafficTypes DO
          IF NOT DisplayNetwork(i,NetworkName) THEN BadFile := TRUE
      END(*CASE 'a'*);
      'q','Q':
      BEGIN

      (*Make sure that original data is back in memory*)

       IF TrafficIndex <> 0 THEN
```

```
        IF RetrieveNetwork(TrafficIndex, NetworkName) THEN ;
       command := 'q'
      END(*CASE 'q'*)
    END(*CASE command/Display menu*);
    IF BadFile THEN
    BEGIN
      BEEP;
      TextAttr := TextAttr OR BlinkOn;
      WriteLn;
      Write(' The specified data file cannot be opened:');
      Write(' press any key to continue.');
      ReadLn;
      TextAttr := TextAttr AND BlinkOff;
      ClrScr
    END(*IF BadFile*)
  END(*WHILE command ...*)
END(*CASE Display*);
'R','r':
BEGIN
  NewScreen('MAIN/Retrieve');
  MainWindow;
  WriteLn;
  Write(' Enter disk file name for network: ');
  ReadLn(NetworkName);
  Write(' Enter traffic type of interest: ');
  ReadLn(TrafficIndex);
  WriteLn;
  IF NOT RetrieveNetwork(TrafficIndex, NetworkName) THEN
    Write(' Retrieval from disk failed: ')
  ELSE
  BEGIN
    Write(' Network data loaded to memory: ');
    NetDefined :=TRUE
  END(*IF NOT RetrieveNetwork... ELSE...*);
  Write('press any key to exit to MAIN Menu.');
  ReadLn
END(*CASE Retrieve*);
'T','t':
BEGIN
  REPEAT
    NewScreen('MAIN/Thruputs');
    MainWindow;
    Write(' C(ompute,  D(isplay,  P(rint,  Q(uit: ');
    ReadLn(command);
    CASE command OF
      'C','c':
      BEGIN
        NewScreen('MAIN/Thruputs/Compute');
        MainWindow;
        Write('Compute A(ll or a S(pecific throughput? ');
        ReadLn(command);
        CASE command OF
          'A','a':
          BEGIN
            FOR i := 1 TO NumberStations DO
              IF NOT SolveThruputs(i, NetworkName) THEN
              BEGIN
                BEEP;
                STR(i:3,message);
```

```
            message := 'Thruput computation failed for traffic type'
                    + message;
            CenterText(message)
          END
          ELSE Write('Thruput computed for traffic type ',i:3);
        Write(': press any key')
      END(*CASE 'A'*);
      'S','s':
      BEGIN
        Write('Enter traffic type for which to compute throughputs: ');
        ReadLn(i);
        IF NOT SolveThruputs(i, NetworkName) THEN
        BEGIN
          BEEP;
          Write('Solution for throughputs failed:')
          END
        ELSE Write('Throughputs computed:');
        Write(' press any key to continue. ');
        ReadLn
      END(*CASE 'S'*)
    END(*CASE command...*)
  END(*CASE 'C'*);
  'D','d':
  BEGIN
    WriteLn(' Function not implemented: press any key to continue.');
    ReadLn
  END;
  'P','p':
  BEGIN
    WriteLn(' Function not implemented: press any key to continue.');
    ReadLn
  END;
  'Q','q': command := 'q';
 END(*MAIN/Thruput CASES*);
UNTIL command = 'q';
END(*CASE 'T'*);
'P','p':
BEGIN
 REPEAT
  NewScreen('MAIN/Paths');
  MainWindow;
  Write(' C(reate, A(dd, D(elete, V(erify, H(ardcopy, Q(uit: ');
  ReadLn(command);
  CASE command OF
    'C','c':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
    'A','a':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
    'D','d':
    BEGIN
      WriteLn(' Function not implemented: press any key to continue.');
      ReadLn
    END;
```

```
          'V','v':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'H','h':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'Q','q': command := 'q'
        END(*CASE command...*)
      UNTIL command = 'q'
    END(*CASE Paths*);
    'M', 'm':
    BEGIN
      REPEAT
        NewScreen('MAIN/Metrics');
        MainWindow;
        Write(' N(odes,  P(aths, Q(ueue length density,  H(ardcopy,  E(nd: ');
        ReadLn(command);
        CASE command OF
          'N','n':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'P','p':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'Q','q':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'H','h':
          BEGIN
            WriteLn(' Function not implemented: press any key to continue.');
            ReadLn
          END;
          'E','e': command := 'e'
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE Metrics*);
    'Q','q':  quit := TRUE
  END(*MAIN Menu CASES*)
UNTIL quit
END(*Main Program*).
```

```
UNIT Data_IO;                (*John R. Doner  20 July 1989*)

(*************************************************************************
  This unit supplies all of the procedures, data types and variables
  needed to manage the input data associated with a full network
  description as required by the AIRMICS MultiMedia Network Design
  Closed-Form Queueing Model.
*************************************************************************)

INTERFACE

  USES DOS, CRT;

  CONST AnyFile = $3F;          (*Used by DOS FindFirst() procedure.*)
        BlinkOn = BYTE(132);    (*Used to blink warning messages*)
        BlinkOff = BYTE(123);   (*Turn off blinking*)
        MaxNodes = 30;          (*Maximum number of nodes: memory limited.*)
        MaxMediumTypes = 3;     (*Maximum number of media types.*)
        MaxTrafficTypes = 3;    (*Maximum number of traffic types.*)
        FormFeed = CHR(12);     (*Formfeed control code for printer.*)

(*************************************************************************
          DICTIONARY OF SIGNIFICANT PROGRAM VARIABLES AND TYPES

  AckLength             -- length of the acknowledgement for current traffic
  ErrorRates[i]         -- missed message rates (MMR) of current traffic type
                           relative to each medium
  GlobalArrivalRate     -- network-wide arrival rate of current traffic type
  MediumBandWidth[i]    -- bandwidths (bits/second) of the available media
  NumberMediumTypes     -- total number of media available in the network
  NumberStations        -- total number communications nodes in network
  NumberTrafficTypes    -- total number of traffic types in system
  StationMultiplex[i,j] -- vectors used to media multiplex traffic at stations
  StationSourceRate[i]  -- relative traffic origination rate for station i
  StationThruPuts[i,j]  -- the relative station throughputs for each traffic
                           type (calculated from input data)
  StoredData            -- a type used by the Fetch() procedure to determine
                           which type of data is to be copied to the current
                           data input process from previously stored data
  Topology[i,j]         - traffic routing transition probability matrix
  TrafficLength         - length of the traffic type currently being considered

*************************************************************************)

  TYPE StoredData = (errors, arrivals, multiplex, connectivity);

  VAR NumberStations,
      NumberMediumTypes,
      NumberTrafficTypes     : INTEGER;
      Topology               : ARRAY[1..MaxNodes, 0..MaxNodes] OF REAL;
      StationMultiplex       : ARRAY[1..MaxNodes, 1..MaxMediumTypes] OF REAL;
      GlobalArrivalRate,
      TrafficLength,
      AckLength              : REAL;
      StationSourceRate      : ARRAY[1..MaxNodes] OF PEAL;
      StationThruputs        : ARRAY[1..MaxTrafficTypes, 1..MaxNodes] OF REAL;
      MediumBandWidth        : ARRAY[1..MaxMediumTypes] OF REAL;
      ErrorRates             : ARRAY[1..MaxMediumTypes] OF REAL;
```

```
(*************************************************************************
  The following procedure formats and centers the string variable passed to it
  and writes it to the screen.
*************************************************************************)

  PROCEDURE CenterText(message: STRING);


(*************************************************************************
  The following procedure emits a short tone from the speaker to alert
  the user to warning messages on the screen.
*************************************************************************)

  PROCEDURE BEEP;


(*************************************************************************
  The following function adds ".top" to the input filename, and then
  retrieves the data from the so-named disk file containing a network
  description. "TrafficIndex" designates for which traffic type the data is
  to be retrieved. The input file should be closed when this procedure is
  entered, and will be closed at procedure exit. TRUE is returned only if
  the retrieve operation is successful.
*************************************************************************)

  FUNCTION RetrieveNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
  The following function stores the computed relative throughput data to
  the file named by the input FileName, after adding the extension ".thp"
  to the filename.  TRUE is returned only if the store operation was
  successful. NOTE: the throughputs should be stored under the same filename
  as is the network data.  The two files will have the same base name, but
  extensions ".top" for the network data and ".thp" for the throughput data
  The output file should be closed when the procedure is called, and will be
  closed at procedure exit.
*************************************************************************)

  FUNCTION StoreThruPuts(FileName: STRING) BOOLEAN;


(*************************************************************************
  The following function retrieves throughput data from a disk file,
  after adding the extension ".thp" to the input filename.  TRUE is returned
  only if the operation is successful. The file should be closed upon
  entry to the procedure, and will be closed at procedure exit.
*************************************************************************)

  FUNCTION RetrieveThruPuts(FileName: STRING) BOOLEAN;


(*************************************************************************
  The following procedure locates specific data fields within the "DataFile"
  file and retrieves them, writing them into the analogous program variables
  representing the data type retrieved.  Any previous value of that data type
  extant in memory is overwritten.  The data retrieved is of the type
  requested by the input "DataType", and the specific instantiation retrieved
  is that associated with the traffic type denoted by "TrafficIndex" or
```

```
  "TrafficIndex" and "station". For fetches of all data types except station
  multiplex data, the data returned is that associated with a previously
  defined traffic type: "station" is ignored during such a request. For
  station multiplex data, the data type returned is for the current traffic
  type and a previous station. TRUE is returned only if the data retrieval
  was successful. Fetch neither opens nor closes the data file, and leaves
  the file pointer at its original position upon exit.
*****************************************************************************)

  PROCEDURE FetchData(TrafficIndex, station: INTEGER; DataType: StoredData;
                      VAR DataFile: FILE);


(*****************************************************************************
  The following function checks for three required types of consistency
  in the current data. First, it checks that the rows of the Topology matrix
  for the current traffic type sum to one. Then it checks to see that the
  sum of the StationSourceRates is one, and finally it checks to see that
  no medium at any station is required to carry more than 100% of its bandwidth
  in traffic as a result of the media multiplexing scheme. Due to the
  inexactness of digital computation, the first two checks actually only
  require that the sums be within 1% of 1.0. When that is the case, the
  summed values are normalized to obtain the maximum precision available
  within the limits of the type REAL floating point number format. FALSE is
  returned if any constraint is not met, and an internal message is written
  to the screen indicating the nature of the inconsistency. The data file
  should be closed upon procedure entry, and will be closed at procedure exit.
  The input variable "HardCopy", when true cause printout of the verify data
  to the printer.
*****************************************************************************)

  FUNCTION Verify(FileName: STRING; HardCopy: BOOLEAN): BOOLEAN;


(*****************************************************************************
  The following procedure prompts the user for all input data required for
  the definition of a communications network. A disk file is used for output
  of the data. The file, if already in existence, should be closed before
  procedure entry, and will be closed at procedure exit.
*****************************************************************************)

  PROCEDURE CreateNetwork(FileName: STRING);


(*****************************************************************************
  The following procedure prompts the user for desired changes to the net-
  work. Use of this procedure is predicated on the existence of an already
  defined network in the current directory, under the input name "FileName".
  The EditNetwork procedure makes a copy of that network file on disk, and
  makes all alterations to the copy. At the end of the edit session, the
  user may choose whether the copy is to replace the original, or is to be
  stored under a separate name. The original file to be edited should be
  closed at procedure entry, and will be closed at procedure exit.
*****************************************************************************)

  PROCEDURE EditNetwork(FileName: STRING);
```

```
(*************************************************************************
  The following procedure provides a hardcopy output of all the input data
  required to define a network and a single traffic type. Entry of a "0" as
  the traffic index results in display of the network wide variables. For
  all entrys of legitimate values of "TrafficIndex", the information specific
  to that traffic type will be printed. The file to be used should be closed
  upon procedure entry, and will be closed at procedure exit. TRUE is
  returned only if the file can be opened.
*************************************************************************)

  FUNCTION DisplayNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
  The following procedure displays the absolute messsage throughputs of the
  nodes for traffic type designated by "TrafficIndex". The filename should
  be the same as that under which the network topology data was stored.
  The file should be closed upon procedure entry, and will be closed at
  procedure exit. TRUE is returned only if the file is found and successfully
  opened.
*************************************************************************)

  FUNCTION DisplayThruputs(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;


(*************************************************************************
*************************************************************************)
IMPLEMENTATION

  PROCEDURE CenterText(message: STRING);
  VAR spaces: INTEGER;

  BEGIN
    spaces := (69 - Length(message)) DIV 2;
    message := '**** ' + message + ' ****';
    WriteLn(' ':spaces, message)
  END(*CenterText*);

  (*The following procedure gets the global data needed to size the file. This
    procedure is not available to calling programs.*)

  PROCEDURE GetGlobal(VAR DataFile: FILE);
  VAR i: INTEGER;
  BEGIN
    SEEK(DataFile, 0);
    Blockread(DataFile, Numberstations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes))
  END(*GetGlobal*);


  PROCEDURE BEEP;
  BEGIN
    SOUND(1000);
    DELAY(200);
    NOSOUND
  END;
```

```
FUNCTION RetrieveNetwork(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;
VAR i, j: INTEGER;
    StartPoint, PreLoop, LoopSize: LONGINT;
    DataFile: FILE;
    FileInfo: SearchRec;

BEGIN
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    RetrieveNetwork := FALSE;
    EXIT
  END
  ELSE
  BEGIN
    ASSIGN(DataFile, FileName);
    RESET(DataFile, 1);
    BlockRead(DataFile, NumberStations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
    IF TrafficIndex > NumberTrafficTypes THEN
    BEGIN
      Write('This traffic type does not exist in ', FileName,': ');
      Write('press any key to continue.');
      ReadLn;
      CLOSE(DataFile);
      RetrieveNetwork := FALSE;
      EXIT
    END(*IF TrafficIndex > ...*);
    PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
    LoopSize := (3 + NumberStations + (NumberStations + 1)*NumberMediumTypes
              + SQR(NumberStations))*SIZEOF(REAL);
    StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
    SEEK(DataFile, StartPoint);
    BlockRead(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
    BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
    BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
    BlockRead(DataFile, AckLength, SIZEOF(AckLength));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, ErrorRates[i], SIZEOF(ErrorRates[i]));
    FOR i := 1 TO NumberStations DO
      BlockRead(DataFile, StationSourceRate[i], SIZEOF(StationSourceRate[i]));
    FOR i := 1 TO NumberStations DO
      FOR j := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, StationMultiplex[i,j], SIZEOF(StationMultiplex[i,j]));
    FOR i := 1 TO NumberStations DO
      FOR j := 0 TO NumberStations DO
        BlockRead(DataFile, Topology[i,j], SIZEOF(Topology[i,j]));
    RetrieveNetwork := TRUE;
    CLOSE(DataFile)
  END(*IF DOSError...ELSE...*)
END(*RetrieveNetwork*);
```

```
  FUNCTION StoreThruputs(FileName: STRING): BOOLEAN;
  VAR i, j: INTEGER;
      ThruputFile: FILE;

  BEGIN
    FileName := FileName + '.thp';
    ASSIGN(ThruputFile, FileName);
{$I-}
    REWRITE(ThruputFile, 1);
{$I+}
    IF IOResult <> 0 THEN
    BEGIN
      Write('Throughput storage file could not be opened: press any key to');
      WriteLn(' continue.');
      ReadLn;
      StoreThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN

     (*Insert the following data to make the file self-contained.*)

      BlockWrite(ThruputFile, NumberStations, SIZEOF(NumberStations));
      BlockWrite(ThruputFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));

      FOR i := 1 TO NumberTrafficTypes DO
        FOR j := 1 TO NumberStations DO
          BlockWrite(ThruputFile, StationThruputs[i, j],
            SIZEOF(StationThruputs[i,j]));
      StoreThruPuts := TRUE;
      CLOSE(ThruputFile)
    END(*IF IOResult... ELSE...*)
  END(*StoreThruputs*);

  FUNCTION RetrieveThruputs(FileName: STRING): BOOLEAN;
  VAR i, j: INTEGER;
      ThruputFile: FILE;
      FileInfo: SearchRec;

  BEGIN
    FileName := FileName + '.thp';
    FindFirst(FileName, AnyFile, FileInfo);
    IF DOSError <> 0 THEN
    BEGIN
      Write('Throughput file ', Filename,' not found: press any key to ');
      WriteLn('continue.');
      ReadLn;
      RetrieveThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      ASSIGN(ThruputFile, FileName);
{$I-}
      REWRITE(ThruputFile, 1);
{$I+}
      IF IOResult <> 0 THEN
      BEGIN
```

```
      Write('Throughput storage file could not be opened: press any key to');
      WriteLn(' continue.');
      ReadLn;
      RetrieveThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      BlockRead(ThruputFile, NumberStations, SIZEOF(NumberStations));
      BlockRead(ThruputFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
      FOR i := 1 TO NumberTrafficTypes DO
        FOR j := 1 TO NumberStations DO
          BlockWrite(ThruputFile, StationThruputs[i, j],
                     SIZEOF(StationThruputs[i,j]));
      RetrieveThruputs := TRUE;
      CLOSE(ThruputFile)
    END(*IF IOResult...ELSE...*)
  END(*IF DOSError ... ELSE...*)
END(*RetrieveThruputs*);

PROCEDURE FetchData(TrafficIndex, station: INTEGER; DataType:
                    StoredData; VAR DataFile: FILE);

VAR i, j: INTEGER;
    FileStart, PreLoop, LoopSize, InLoop, StartPoint: LONGINT;

BEGIN
  FileStart := FilePos(DataFile);
  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations + SQR(NumberStations) +
               NumberStations*NumberMediumTypes)*SIZEOF(REAL);
  StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
  CASE DataType OF
    errors:
    BEGIN
      InLoop := StartPoint + 3*SIZEOF(REAL);
      Seek(DataFile, InLoop);
      FOR i := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, ErrorRates[i], SIZEOF(ErrorRates[i]))
    END(*errors*);
    arrivals:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes)*SIZEOF(REAL);
      Seek(DataFile, InLoop);
      FOR i := 1 TO NumberStations DO
        BlockRead(DataFile, StationSourceRate[i], SIZEOF(StationSourceRate[i]))
    END(*arrivals*);
    multiplex:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes + NumberStations +
                (station - 1)*NumberMediumTypes)*SIZEOF(REAL);
      SEEK(DataFile, InLoop);
      FOR j := 1 TO NumberMediumTypes DO
        BlockRead(DataFile, StationMultiplex[station,j],
                  SIZEOF(StationMultiplex[station,j]))
    END(*multiplex*);
    connectivity:
    BEGIN
      InLoop := StartPoint + (3 + NumberMediumTypes
```

```
            + NumberStations*(NumberMediumTypes + 1))*SIZEOF(REAL);
      SEEK(DataFile, InLoop);
      FOR i := 1 TO NumberStations DO
        FOR j := 0 TO NumberStations DO
          BlockRead(DataFile, Topology[i,j], SIZEOF(REAL));
    END(*connectivity*)
  END(*CASES*);
  Seek(DataFile, FileStart)
END(*FetchData*);

FUNCTION Verify(FileName: STRING; HardCopy: BOOLEAN): BOOLEAN;
VAR i, j, k, m: INTEGER;
    sum: REAL;
    message, message2: STRING;
    Transitions, Capacity, SourceRates: BOOLEAN;
    DataFile: FILE;
    FileInfo: SearchRec;
    MultiplexSums: ARRAY[1..MaxNodes,1..MaxMediumTypes] OF REAL;
    lst: TEXT;

BEGIN
(*Open file.*)

  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN

  (*Can't open, so make a graceful exit.*)

    Write('File ', FileName, ' not found: press any key to continue.');
    ReadLn;
    EXIT
  END
  ELSE
  BEGIN
    IF HardCopy THEN
    BEGIN
      ASSIGN(lst, 'prn');
      REWRITE(lst)
    END;

    (*Open file, and get the essential parameters at the top of file.*)

    ASSIGN(DataFile, FileName);
    RESET(DataFile, 1);
    BlockRead(DataFile, NumberStations, SIZEOF(NumberStations));
    BlockRead(DataFile, NumberMediumTypes, SIZEOF(NumberMediumTypes));
    FOR i := 1 TO NumberMediumTypes DO
      BlockRead(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
    BlockRead(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));

    IF HardCopy THEN
    BEGIN
      WriteLn(lst, 'Verification of routing transition probabilities:');
      WriteLn(lst)
    END;
    Transitions := TRUE;
    FOR i := 1 TO NumberTrafficTypes DO
```

```
  BEGIN

(*First check for consistency of topology information.*)

    FetchData(i, 0, connectivity, DataFile);
    FOR j := 1 TO NumberStations DO
    BEGIN
      sum := 0.0;
      FOR k := 0 TO NumberStations DO sum := sum + Topology[j, k];
      IF ABS(sum - 1.0) <= 0.01 THEN
        FOR m := 0 TO NumberStations DO Topology[i,m] := Topology[i,m]/sum
      ELSE
      BEGIN
        Transitions := FALSE;
        Str(i:3, message);
        message := 'WARNING: routing probability data, traffic type '
          + message + ' is inconsistent.';
        CenterText(message);
        IF HardCopy  THEN
        BEGIN
          Write(lst, '  Routing probability data for traffic type ',i:3);
          WriteLn(lst, ', station ',j:3, ' sums to ', sum:5:3)
        END
      END;
    END(*FOR j ...*)
  END(*FOR i...*);
  WriteLn;
  IF (HardCopy AND Transitions) THEN
    WriteLn(lst, 'No inconsistencies were found.');

(*Next, check for consistency of source rate data.*)

  IF HardCopy THEN
  BEGIN
    WriteLn(lst);
    WriteLn(lst);
    WriteLn(lst, 'Verification of station arrival rate data:');
    WriteLn(lst)
  END;
  SourceRates := TRUE;
  FOR i := 1 TO NumberTrafficTypes DO
  BEGIN
    sum := 0.0;
    FetchData(i, 0, arrivals, DataFile);
    FOR j := 1 TO NumberStations DO  sum := sum + StationSourceRate[j];
    IF ABS(sum - 1.0) <= 0.01 THEN
      FOR j := 1 TO NumberStations DO
        StationSourceRate[j] := StationSourceRate[j]/sum
    ELSE
    BEGIN
      SourceRates := FALSE;
      Str(i:3, message);
      message := 'WARNING: station arrival rates for traffic type'
        +  message + ' are inconsistent ';
      CenterText(message);
      IF HardCopy THEN
      BEGIN
        Write(lst, '  Station arrival rates for traffic type ', i:3);
        WriteLn(lst, ' sum to ',sum:5:2)
```

```
      END
    END(*FOR j ...*)
  END(*FOR i...*);
  WriteIn;
  IF (HardCopy AND SourceRates) THEN
    WriteLn(lst, 'No inconsistencies were found.');

 (*Finally, sum up the traffic type demands on the available channels.*)

  IF HardCopy THEN
  BEGIN
    WriteLn(lst);
    WriteLn(lst);
    WriteLn(lst, 'Verification of channel capacity constraints: ');
    WriteLn(lst)
  END;
  Capacity := TRUE;
  FOR i := 1 TO NumberStations DO
    FOR j := 1 TO NumberMediumTypes DO MultiPlexSums[i,j] := 0.0;
  FOR i := 1 TO NumberTrafficTypes DO
    FOR j := 1 TO NumberStations DO
    BEGIN
      FetchData(i, j, multiplex, DataFile);
      FOR k := 1 TO NumberMediumTypes DO
        MultiplexSums[j,k] := MultiplexSums[j,k] + StationMultiplex[j, k]
    END(*FOR i, j, k...*);
    FOR j := 1 TO NumberStations DO
      FOR k := 1 TO NumberMediumTypes DO
        IF MultiplexSums[j, k] > 1.0 THEN
        BEGIN
          STR(k:3, message);
          STR(j:3, message2);
          message := 'WARNING: channel capacity for medium ' + message +
            ', station ' + message2 + ' exceeded.';
          CenterText(message);
          IF HardCopy THEN
          BEGIN
            Write(lst, '  Multiplexed channel capacity for medium ',k:3);
            WriteLn(lst,' at station ',j:3, ' sums to ',MultiplexSums[j,k]:5:3)
          END;
          Capacity := FALSE
        END(*FOR i, j, IF...*);
   IF (HardCopy AND Capacity) THEN
    WriteLn(lst, 'No inconsistencies were found.');
  IF HardCopy THEN CLOSE(lst);
  Verify := Transitions AND Capacity AND SourceRates
 END(*IF DOSError...ELSE...*)
END(*Verify*);

PROCEDURE CreateNetwork;
VAR i, j, k, index: INTEGER;
    sum: REAL;
    FullName, message: STRING;
    command: CHAR;
    DataFile: FILE;
    FileInfo: SearchRec;

BEGIN
```

```
(*File name  .ntry*)
  REPEAT
    CenterText('NETWORK CREATION');
    command := 'y';
    WriteLn;
    WriteLn('Data will be stored to disk as it is entered.');
    WriteLn;
    FullName := FileName + '.top';
    FindFirst(FullName, AnyFile, FileInfo);
    IF DOSError = 0 THEN
    BEGIN
      WriteLn;
      BEEP;
      TextAttr := TextAttr OR BlinkOn;
      CenterText('WARNING: Like-named file already on disk will be destroyed.');
      TextAttr := TextAttr AND BlinkOff;
      WriteLn;
      Write('      Proceed anyway? (y/n): ');
      ReadLn(command);
      ClrScr
    END
    ELSE
      command := 'y';
  UNTIL command = 'y';
  ASSIGN(DataFile, FullName);
  REWRITE(DataFile, 1);

(*NumberStations*)
  REPEAT
    Write('Enter number of communications stations (not exceeding ');
    Write(MaxNodes:3,'): ');
    ReadLn(NumberStations)
  UNTIL NumberStations <= MaxNodes ;
  BlockWrite(DataFile, NumberStations, SIZEOF(INTEGER));
  WriteLn;

(*NumberMediaTypes*)
  REPEAT
    Write('Enter number of media types (not exceeding ');
    Write(MaxMediumTypes:3,'): ');
    ReadLn(NumberMediumTypes)
  UNTIL NumberMediumTypes <= MaxMediumTypes;
  BlockWrite(DataFile, NumberMediumTypes, SIZEOF(INTEGER));
  WriteLn;

(*MediumBandWidth[.]*)
  FOR i := 1 TO NumberMediumTypes DO
  BEGIN
    Write('  Enter bandwidth of medium type ', i:3, ' (Kbits/second): ');
    ReadLn(MediumBandWidth[i]);
    BlockWrite(DataFile, MediumBandWidth[i], SIZEOF(MediumBandWidth[i]));
  END;
  WriteLn;

(*NumberTrafficTypes*)
  REPEAT
    Write('Enter number of traffic types (not exceeding ');
    Write(MaxTrafficTypes:3, '): ');
    ReadLn(NumberTrafficTypes)
```

```
  UNTIL NumberTrafficTypes <= MaxTrafficTypes;
  BlockWrite(DataFile, NumberTrafficTypes, SIZEOF(NumberTrafficTypes));
  WriteLn;

(*All remaining data is traffic type dependent, and so will be entered
  for each traffic type.*)

  FOR i := 1 TO NumberTrafficTypes DO
  BEGIN
    ClrScr;
    WriteLn;
    BEEP;
    TextAttr := TextAttr OR BlinkOn;
    Str(i:3, message);
    message := 'Data input for traffic type ' + message;
    CenterText(message);
    TextAttr := TextAttr AND BlinkOff;

(*GlobalArrivalRate*)
    WriteLn;
    Write('Enter the network-wide traffic type arrival rate  (messages/sec.): ');
    ReadLn(GlobalArrivalRate);
    BlockWrite(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
    WriteLn;

(*TrafficLength*)
    Write('Enter mean message length (in bits) for traffic type: ');
    ReadLn(TrafficLength);
    BlockWrite(DataFile, TrafficLength, SIZEOF(TrafficLength));
    WriteLn;

(*AckLength*)
    Write('Enter mean length (bits) for acknowledgement message (0 if none sent): ');
    ReadLn(AckLength);
    BlockWrite(DataFile, AckLength, SIZEOF(AckLength));
    WriteLn;

(*ErrorRates*)
    ClrScr;
    CenterText('Entry of traffic type MMR for each medium type');
    WriteLn;
    Write('Copy previous missed message rates? (n/y): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, errors, DataFile)
    END
    ELSE
      FOR j := 1 TO NumberMediumTypes DO
      BEGIN
        Write('Enter missed message rate for medium ',j:2,': ');
        ReadLn(ErrorRates[j])
      END;
    FOR j := 1 TO NumberMediumTypes DO
      BlockWrite(DataFile, ErrorRates[j], SIZEOF(ErrorRates[j]));
```

```
    WriteLn;

  (*StationSourceRate*)
    ClrScr;
    CenterText('Station relative arrival rates for traffic type');
    WriteLn;
    Write('Copy previous arrival rates? (n/y): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, arrivals, DataFile);
    END
    ELSE
      FOR j := 1 TO NumberStations DO
      BEGIN
        Write('Enter station arrival rate for station ',j:2,': ');
        ReadLn(StationSourceRate[j])
      END;
    FOR j := 1 TO NumberStations DO
      BlockWrite(DataFile, StationSourceRate[j], SIZEOF(StationSourceRate[j]));
    WriteLn;

  (*StationMultiplex*)
    ClrScr;
    CenterText('Entry of traffic type/media type multiplex data');
    WriteLn;
    FOR j := 1 TO NumberStations DO
    BEGIN
      WriteLn('Entry of media multiplex vector for station ',j:3,'.');
      Write('Copy previous multiplex vector? (y/n): ');
      ReadLn(command);
      IF command = 'y' THEN
      BEGIN
        REPEAT
          Write('Enter previously defined station from which to copy: ');
          ReadLn(index)
        UNTIL index < j;
        FetchData(i, index, multiplex, DataFile)
      END
      ELSE
        FOR k := 1 TO NumberMediumTypes DO
        BEGIN
          Write('Enter fraction of medium ', k:2,' dedicated to this traffic: ');
          ReadLn(StationMultiplex[j, k])
        END;
        FOR k := 1 TO NumberMediumTypes DO
          BlockWrite(DataFile, StationMultiplex[j,k], SIZEOF(StationMultiplex[j,k]));
      WriteLn
    END(*FOR j...*);

  (*Topology*)
    ClrScr;
    CenterText('Entry of topology matrix for this traffic type');
    WriteLn;
    WriteLn('Note:  station "0" is the sink for all messages, so enter the ');
```

```
    WriteLn('       proportion of traffic terminating at node i for the ');
    WriteLn('       [i, 0] entry of the topology matrix.');
    WriteLn;
    Write('Copy previous topology matrix? (y/n): ');
    ReadLn(command);
    IF command = 'y' THEN
    BEGIN
      REPEAT
        Write('Enter previously defined traffic type from which to copy: ');
        ReadLn(index)
      UNTIL index < i;
      FetchData(index, 0, connectivity, DataFile)
    END
    ELSE
    BEGIN

      (*Initialize all transition probabilities to zero.*)

      FOR j := 1 TO NumberStations DO
        FOR k := 0 TO NumberStations DO Topology[j, k] := 0.0;
      WriteLn('Enter (origin, destination, probability), with data entries');
      WriteLn(' separated by spaces, followed by a <ENTER>.  To terminate ');
      WriteLn('the process, enter a probability of zero with any node pair.');
      WriteLn;
      REPEAT
        Write('Origin node, Destination Node, Probability --> ');
        ReadLn(j, k, sum);
        IF sum <> 0 THEN Topology[j, k] := sum
      UNTIL  sum = 0.0
    END(*IF command...ELSE...*);
    FOR j := 1 TO NumberStations DO
      FOR k := 0 TO NumberStations DO
        BlockWrite(DataFile, Topology[j,k], SIZEOF(Topology[j,k]));
    WriteLn
  END(*FOR i ...*);
  CLOSE(DataFile);

(*Data verification*)

  ClrScr;
  WriteLn;
  CenterText('DATA VERIFICATION');
  WriteLn;
  IF NOT Verify(FileName, FALSE) THEN
  BEGIN
    BEEP;
    TextAttr := TextAttr OR BlinkOn;
    CenterText('WARNING: this data must be edited before use.');
    TextAttr := TextAttr + BlinkOff
  END
  ELSE
    WriteLn('Data entered does not violate any consistency rule.');
  Write(': press any key to continue.');
  ReadLn
END(*CreateNetwork*);

PROCEDURE EditNetwork(FileName: STRING);
VAR i, j, EditChoice, index: INTEGER;
    command: CHAR;
```

```
    quit: BOOLEAN;
    value, sum: REAL;
    DataFile, TempFile: FILE;
    FileInfo: SearchRec;
    TempFileName, NewName: STRING;
    data: BYTE;
    PreLoop, LoopSize, InLoop, StartPoint: LONGINT;

BEGIN
  quit := FALSE;

(*Create a copy of the input data file on which to make editing changes.*)

  TempFileName := FileName + '.tmp';
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    Write('Cannot find the file to be edited: press any key to continue.');
    ReadLn;
    EXIT
  END;
  ASSIGN(DataFile, FileName);
  RESET(DataFile, 1);
  ASSIGN(TempFile, TempFileName);
  REWRITE(TempFile, 1);
  WHILE NOT EOF(DataFile) DO
  BEGIN
    BlockRead(DataFile, data, 1);
    BlockWrite(TempFile, data, 1)
  END(*WHILE NOT...*);
  GetGlobal(DataFile);          (*Get the required global parameters from the file.*)
  CLOSE(DataFile);

  (*Obtain essential "sizing" parameters for getting around in the file.*)

  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations + SQR(NumberStations) +
               NumberStations*NumberMediumTypes)*SIZEOF(REAL);

  (*Main edit menu follows.*)

  WriteLn;
  REPEAT
    WriteLn(' Edit functions are as follows:');
    WriteLn('   0. exit the edit function,');
    WriteLn('   1. modify media bandwidths,');
    WriteLn('   2. modify traffic type global arrival rates,');
    WriteLn('   3. modify traffic type traffic length,');
    WriteLn('   4. modify traffic type acknowledgement length,');
    WriteLn('   5. modify traffic type/medium type error rates');
    WriteLn('   6. modify traffic type station arrival rates,');
    WriteLn('   7. modify traffic type medium multiplex vector,');
    WriteLn('   8. modify traffic type station connectivity,');
    WriteLn;
    Write(' Enter integer corresponding to choice: ');
    ReadLn(EditChoice);
    CASE EditChoice OF
      0:        (*Exit procedure and saving edited file to disk.*)
```

```
BEGIN
  quit := TRUE;
  CLOSE(TempFile);
  WriteLn;
  Write(' Save as O(riginal, as N(ew, or E(xit without saving?: ');
  ReadLn(command);
  CASE command OF
    'o','O':
    BEGIN
      ERASE(DataFile);
      RENAME(TempFile, FileName)
    END(*CASE 'o'*);
    'n','N':
    BEGIN
      REPEAT
        Write(' Enter filename under which to store edited data: ');
        ReadLn(NewName);
        NewName := NewName + '.top';
        FindFirst(NewName, AnyFile, FileInfo);
        IF DOSError = 0 THEN
        BEGIN
          WriteLn;
          BEEP;
          TextAttr := TextAttr OR BlinkOn;
          CenterText('WARNING: File of that name already exists.');
          TextAttr := TextAttr AND BlinkOff;
          WriteLn;
          Write(' Press any key to continue.');
          ReadLn(command)
        END
        ELSE
        BEGIN
          RENAME(TempFile, NewName);
          WriteLn(' File ', NewName, 'stored to disk.');
        END
      UNTIL DOSError <> 0
    END;
    'e','E': ERASE(TempFile)
  END(*CASE command...*)
END(*CASE 0*);
1:                              (*Modify media bandwidths*)
BEGIN
  REPEAT
    WriteLn;
    Write(' Modify which medium bandwidth?: ');
    ReadLn(index)
  UNTIL index <= NumberMediumTypes;
  StartPoint := 2*SIZEOF(INTEGER)
              + (index - 1)*SIZEOF(REAL);
  SEEK(TempFile, StartPoint);
  BlockRead(TempFile, value, SIZEOF(value));
  WriteLn(' Existing value is ', value:8:2);
  Write(' Modify to what value?: ');
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 1*);
2:                      (*Modify traffic type global arrival rate*)
BEGIN
```

```
    REPEAT
      WriteLn;
      Write(' Modify which traffic type global arrival rate?: ');
      ReadLn(index)
    UNTIL index <= NumberTrafficTypes;
    StartPoint := PreLoop + (index - 1)*LoopSize;
    SEEK(TempFile, StartPoint);
    BlockRead(TempFile, value, SIZEOF(value));
    WriteLn(' Existing value is ', value:8:2);
    Write(' Modify to what value?: ');
    ReadLn(value);
    SEEK(TempFile, StartPoint);
    BlockWrite(TempFile, value, SIZEOF(value))
  END(*CASE 2*);
  3:                (*Modify traffic type traffic length*)
  BEGIN
    REPEAT
      WriteLn;
      Write(' Modify which traffic type traffic length?: ');
      ReadLn(index)
    UNTIL index <= NumberTrafficTypes;
    StartPoint := PreLoop + (index - 1)*LoopSize + SIZEOF(REAL);
    SEEK(TempFile, StartPoint);
    BlockRead(TempFile, value, SIZEOF(value));
    WriteLn(' Existing value is ', value:8:2);
    Write(' Modify to what value?: ');
    ReadLn(value);
    SEEK(TempFile, StartPoint);
    BlockWrite(TempFile, value, SIZEOF(value))
  END(*CASE 3*);
  4:                (*Modify traffic type acknowledgement length*)
  BEGIN
    REPEAT
      WriteLn;
      Write(' Modify which traffic type acknowledgement length?: ');
      ReadLn(index)
    UNTIL index <= NumberTrafficTypes;
    StartPoint := PreLoop + (index - 1)*LoopSize + 2*SIZEOF(REAL);
    SEEK(TempFile, StartPoint);
    BlockRead(TempFile, value, SIZEOF(value));
    WriteLn(' Existing value is ', value:8:2);
    Write(' Modify to what value?: ');
    ReadLn(value);
    SEEK(TempFile, StartPoint);
    BlockWrite(TempFile, value, SIZEOF(value))
  END(*CASE 4*);
  5:          (*Modify traffic type/medium type error rates*)
  BEGIN
    REPEAT
      WriteLn;
      Write(' Modify error rate for which traffic/medium pair?: ');
      ReadLn(index, i)
    UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberMediumTypes));
    StartPoint := PreLoop + (index - 1)*LoopSize +
                  (2 + i)*SIZEOF(REAL);
    SEEK(TempFile, StartPoint);
    BlockRead(TempFile, value, SIZEOF(value));
    WriteLn('Existing value is ', value:6:4);
    Write('Modify to what value?: ');
```

```
  ReadLn(value);
  SEEK(TempFile, StartPoint);
  BlockWrite(TempFile, value, SIZEOF(value))
END(*CASE 5*);
6:              (*Modify station source rates for traffic type*)
BEGIN
  REPEAT
    WriteLn;
    Write(' C(hange station source rate, E(xit (c/e): ');
    ReadLn(command);
    CASE command OF
      'c', 'C':
      BEGIN
        REPEAT
          WriteLn;
          Write(' Modify source rates for which traffic type/');
          Write('station?: ');
          ReadLn(index, i)
        UNTIL ((index <= NumberTrafficTypes) AND (i <=NumberStations));
        StartPoint := PreLoop + (index - 1)*LoopSize
                    + (2 + i + NumberMediumTypes)*SIZEOF(REAL);
        SEEK(TempFile, StartPoint);
        BlockRead(TempFile, value, SIZEOF(value));
        WriteLn(' Existing value is ', value:8:2);
        Write(' Modify to what value?: ');
        ReadLn(value);
        SEEK(TempFile, StartPoint);
        BlockWrite(TempFile, value, SIZEOF(value))
      END(*CASE 'c'*);
      'e','E':
    END(*CASE command...*)
  UNTIL command = 'e'
END(*CASE 6*);
7:              (*Modify traffic type medium multiplex vector*)
BEGIN
  REPEAT
    WriteLn;
    Write(' C(hange a multiplex value, or E(xit (c/e): ');
    ReadLn(command);
    CASE command OF
      'c', 'C':
      BEGIN
        REPEAT
          WriteLn;
          Write(' Modify medium multiplex vector for which traffic');
          Write(' type/ station/ medium?: ');
          ReadLn(index, i, j)
        UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberStations)
              AND(j <= NumberMediumTypes));
        StartPoint := PreLoop + (index - 1)*LoopSize
                    + (3 + NumberMediumTypes + NumberStations
                    + (i - 1)*NumberMediumTypes + j - 1)*SIZEOF(REAL);
        SEEK(TempFile, StartPoint);
        BlockRead(TempFile, value, SIZEOF(value));
        WriteLn(' Existing value is ', value:8:2);
        Write(' Modify to what value?: ');
        ReadLn(value);
        SEEK(TempFile, StartPoint);
        BlockWrite(TempFile, value, SIZEOF(value))
```

```
          END(*CASE 'c'*);
          'e','E':
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE 7*);
    8:
    BEGIN
      REPEAT
        WriteLn;
        Write(' C(hange a transition probability, or E(xit (c/e): ');
        ReadLn(command);
        CASE command OF
          'c', 'C':
          BEGIN
            REPEAT
              WriteLn;
              Write('Modify topology for which traffic type/ origin');
              Write('/destination stations?: ');
              ReadLn(index)
            UNTIL ((index <= NumberTrafficTypes) AND (i <= NumberStations)
                  AND (j <= NumberStations));
            StartPoint := PreLoop + (index - 1)*LoopSize
            + (3 + NumberMediumTypes + NumberStations*(NumberMediumTypes + 1)
            +(i - 1)*NumberStations + j)* SIZEOF(REAL);
            SEEK(TempFile, StartPoint);
            BlockRead(TempFile, value, SIZEOF(value));
            WriteLn(' Existing value is ', value:8:2);
            Write(' Modify to what value?: ');
            ReadLn(value);
            SEEK(TempFile, StartPoint);
            BlockWrite(TempFile, value, SIZEOF(value))
          END(*CASE 'c'*);
          'e','E':
        END(*CASE command...*)
      UNTIL command = 'e'
    END(*CASE 8*)
  END(*CASES*)
UNTIL quit
END(*EditNetwork*);

FUNCTION DisplayNetwork(TrafficIndex: INTEGER, FileName: STRING):BOOLEAN;
VAR i, j: INTEGER;
    PreLoop, LoopSize, StartPoint: LONGINT,
    DataFile: FILE;
    FileInfo: SearchRec;
    lst: TEXT;



BEGIN
  FileName := FileName + '.top';
  FindFirst(FileName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    DisplayNetwork := FALSE;
    EXIT
  END
  ELSE
  BEGIN
```

```
ASSIGN(DataFile, FileName);
RESET(DataFile, 1);
ASSIGN(lst, 'prn');
REWRITE(lst);
IF TrafficIndex = 0 THEN
BEGIN

(*print out the global information.*)

  GetGlobal(DataFile);
  WriteLn(lst, 'GLOBAL DATA FOR NETWORK DEFINITION IN ', FileName);
  WriteLn(lst);
  WriteLn(lst, 'Number of network nodes = ', NumberStations:3);
  WriteLn(lst, 'Number of medium types  = ', NumberMediumTypes:3);
  WriteLn(lst, 'Number of traffic types = ', NumberTrafficTypes:3);
  WriteLn(lst);
  WriteLn(lst, 'Media Bandwidths:');
  FOR i := 1 TO NumberMediumTypes DO
  BEGIN
    Write(lst, 'Bandwidth for medium ', i:3, ' = ');
    WriteLn(lst, MediumBandWidth[i]:8:2,' KBits/sec.')
  END;
  WriteLn(lst, FormFeed)
END
ELSE
BEGIN

(*Print out the specific information concerning a given traffic type.*)

  GetGlobal(DataFile);
  PreLoop := 3*SIZEOF(INTEGER) + NumberMediumTypes*SIZEOF(REAL);
  LoopSize := (3 + NumberMediumTypes + 2*NumberStations +
    SQR(NumberStations) + NumberStations*NumberMediumTypes)*SIZEOF(REAL);
  StartPoint := PreLoop + (TrafficIndex - 1)*LoopSize;
  SEEK(DataFile, StartPoint);

(*global arrival rate, message length, acknowledgement length*)

  BlockRead(DataFile, GlobalArrivalRate, SIZEOF(GlobalArrivalRate));
  BlockRead(DataFile, TrafficLength, SIZEOF(TrafficLength));
  BlockRead(DataFile, AckLength, SIZEOF(AckLength));
  Write(lst, 'INFORMATION FOR TRAFFIC TYPE ',TrafficIndex:3);
  WriteLn(lst ,' IN FILE ', FileName);
  WriteLn(lst);
  Write(lst, 'Traffic type global arrival rate     = ');
  WriteLn(lst, GlobalArrivalRate:9:2, ' messages/sec.');
  Write(lst, 'Traffic type message length          = ');
  WriteLn(lst, TrafficLength:8:1,' bits.');
  Write(lst, 'Traffic type acknowledgement length = ');
  WriteLn(lst, AckLength:8:1, ' bits.');
  WriteLn(lst);

(*missed message rates for all medium types.*)

  FetchData(TrafficIndex, 0, errors, DataFile);
  WriteLn(lst, 'Traffic type missed message rates for the medium types:');
  FOR i := 1 TO NumberMediumTypes DO
    WriteLn(lst, 'MMR for medium type ',i:3, ' = ',ErrorRates[i]:5:3);
  WriteLn(lst);
```

```
    (*source rates for traffic type at each station*)

      FetchData(TrafficIndex, 0, arrivals, DataFile);
      WriteLn(lst, 'Arrival rate for this traffic type at each station:');
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst,'Arrival rate at station ', i:3,' = ');
        WriteLn(lst,StationSourceRate[i]:5:4,' messages/sec.')
      END;
      Write(lst, FormFeed);

    (*station multiplex vectors*)
      FOR j := 1 TO NumberStations DO
        FetchData(TrafficIndex, j, multiplex, DataFile);
      WriteLn(lst, 'Multiplex vectors for traffic type ', TrafficIndex:3);
      WriteLn(lst);
      WriteLn(lst,' ':20,'Medium 1     Medium 2     Medium 3');
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst,'Station ',i:3, '     ':9);
        FOR j := 1 TO NumberMediumTypes DO
          Write(lst,StationMultiplex[i,j]:5:3, ' ':8);
        WriteLn(lst)
      END;
      Write(lst, FormFeed);

  (*traffic type topology matrix*)

      FetchData(TrafficIndex, 0, connectivity, DataFile);
      Write(lst, 'Routing transition probabilities for traffic type ');
      WriteLn(lst, TrafficIndex:3);
      WriteLn(lst, '(The "0-th" entry represents traffic absorption at station)');
      WriteLn(lst);
      Write(lst, ' ':5);
      FOR i := 0 TO NumberStations DO Write(lst, i:3, ' ':5);
      WriteLn(lst);
      FOR i := 1 TO NumberStations DO
      BEGIN
        Write(lst, i:2, '   ');
        FOR j := 0 TO NumberStations DO Write(lst, Topology[i,j]:5:3, ' ':3);
        WriteLn(lst)
      END(*FOR i...*);
      Write(lst, FormFeed)
    END(*IF TrafficIndex...ELSE...*);
    CLOSE(lst);
    CLOSE(DataFile);
    DisplayNetwork := TRUE
  END(*IF DOSError...ELSE...*)
END(*DisplayNetwork*);

FUNCTION DisplayThruputs(TrafficIndex: INTEGER; FileName: STRING): BOOLEAN;
VAR i: INTEGER;
    ThruputFile: FILE;
    StartPoint: LONGINT;
    value: REAL;
    FileInfo: SearchRec;
    lst: TEXT;
```

```
  BEGIN
    FileName := FileName + '.thp';
    FindFirst(FileName, AnyFile, FileInfo);
    IF DOSError <> 0 THEN
    BEGIN
      Write('Throughput file ', FileName,' could not be found:');
      WriteLn('press any key to continue.');
      ReadLn;
      DisplayThruputs := FALSE;
      EXIT
    END
    ELSE
    BEGIN
      ASSIGN(lst, 'prn');
      REWRITE(lst);
      WriteLn(lst, 'THROUGHPUT DATA FOR TRAFFIC TYPE ',TrafficIndex:3);
      WriteLn(lst);
      ASSIGN(ThruPutFile, FileName);
      RESET(ThruputFile, 1);
      BlockRead(ThruputFile, NumberStations, SIZEOF(NumberStations));
      StartPoint := 2*SIZEOF(INTEGER)
                  + (TrafficIndex - 1)*NumberStations*SIZEOF(REAL);
      SEEK(ThruputFile, StartPoint);
      FOR i := 1 TO NumberStations DO
      BEGIN
        BlockRead(ThruputFile, value, SIZEOF(value));
        WriteLn(lst, 'Station ', i:3, ' throughput = ',value:5:3)
      END;
      Write(lst, FormFeed);
      CLOSE(ThruputFile);
      CLOSE(lst);
      DisplayThruputs := TRUE
    END(*IF DOSError...ELSE...*);
  END(*DisplayThruputs*);

BEGIN

END(*Data_IO*).
```

```
UNIT NetComp;                    (*John R. Doner, 3 September 1989*)

(*This unit contains the necessary computional proccedures to determine the
  network relative throughputs, and all derived network performance measures
  associated with the AIRMICS Multimedia Network Design Program (NetCalc).*)


INTERFACE

USES DOS, Data_IO;


(**************************************************************************
  The following procedure solves the required linear system of equations to
  obtain the network relative throughputs for a given traffic type.  Input
  to the procedure is the traffic type and the network file name containing
  the network data, and the output solution is then placed in the appropriate
  row of the StationThruput[ ] array (see Data_IO unit for definition.).
  FALSE is returned only if the equations were found to be non-solvable (i.e.,
  singular matrix).  The file containing network data should be closed at
  entry to this procedure, and will be closed at exit.  Note that this
  function does not store the computed throughputs to disk.
**************************************************************************)

  FUNCTION SolveThruputs(TrafficIndex: INTEGER; NetworkName: STRING): BOOLEAN;

IMPLEMENTATION

  FUNCTION SolveThruputs(TrafficIndex: INTEGER; NetworkName: STRING): BOOLEAN;
  VAR InvertArray: ARRAY[1..MaxNodes,1..MaxNodes + 1] OF REAL;
      TransposeCount, i,j, k: INTEGER;
      divisor, multiplier, temp: REAL;
      transpose: ARRAY[1..MaxNodes,0..1] OF INTEGER;
      FileInfo: SearchRec;
      DataFile: FILE;

  (*"InvertArray[]" holds the enhanced matrix while elementary row operations
    are performed.  "transpose" holds information on any row interchanges
    required during the upper triangularization process. *)

   (*The following function interchanges two rows of the InvertArray matrix if
     that is required to bring a non-zero into a diagonal position. The
     function returns FALSE only if there are no non-zero elements below the
     diagonal.*)

      FUNCTION Interchange(row: INTEGER)  BOOLEAN;
      VAR i, j, k: INTEGER;
          temp : REAL;

      BEGIN
        j := i + 1;
        WHILE ((j < NumberStations + 1) AND (InvertArray[j,i] = 0)) DO
          j := j + 1;
        IF j = NumberStations + 1 THEN
        BEGIN
          Interchange := FALSE;
          EXIT
        END;
```

```
      FOR k := i TO NumberStations + 1 DO
      BEGIN
        temp := InvertArray[i,k];
        InvertArray[i,k] := InvertArray[j,k];
        InvertArray[j,k] := temp
      END(*FOR k...*);
      TransposeCount := TransposeCount + 1;
      transpose[TransposeCount, 0] := i;
      transpose[TransposeCount, 1] := j;
      Interchange := TRUE
    END(*Interchange*);


BEGIN
(*First, open the file and fetch the relaevant data.*)

  NetworkName := NetworkName + '.top';
  FindFirst(NetworkName, AnyFile, FileInfo);
  IF DOSError <> 0 THEN
  BEGIN
    SolveThruputs := FALSE;
    EXIT
  END
  ELSE
  BEGIN
    ASSIGN(DataFile, NetworkName);
    RESET(DataFile, 1);

(*Solution for the thruoughputs is carried out by enhancing the coefficient
  matrix with the column of source rates for the nodes, and then transform-
  ing the coefficient matrix to upper triangular form. From this form, the
  unknowns (thruoughputs), can be iteratively determined from the last to
  the first  (Biblical method).*)

(*First load required data to InvertArray*)

    FetchData(TrafficIndex, 0, arrivals, DataFile);
    FOR i := 1 TO NumberStations DO
      InvertArray[i, NumberStations + 1] := StationSourceRate[i];
    FetchData(TrafficIndex, 0, connectivity, DataFile);
    FOR i := 1 TO NumberStations DO
      FOR j := 1 TO NumberStations DO
        InvertArray[i,j] := -Topology[j,i];
    CLOSE(DataFile)
  END(*IF DOSError...ELSE...*);

  FOR i := 1 TO NumberStations DO
    InvertArray[i,i] := InvertArray[i,i] + 1.0;
  TransposeCount := 0;

  (*Matrix is defined: ready to begin upper triangulation.*)

  FOR i := 1 TO NumberStations -1 DO
  BEGIN

  (*First, check that next diagonal element is non-zero, and perform a
    row transposition if necessary.*)

    IF InvertArray[i,i] = 0 THEN
```

```
        IF NOT Interchange(i) THEN
        BEGIN
          SolveThruputs := FALSE;
          EXIT
        END(*IF NOT ...*);
      divisor := InvertArray[i, i];

      (*Normalize i-th row so diagonal element = 1.*)

      FOR j := i TO NumberStations DO
        InvertArray[i,j] := InvertArray[i,j]/divisor;

      (*Now do subtraction of multiple of i-th row from each following row
        to zero out i-th column below diagonal.*)

      FOR j := i + 1 TO NumberStations DO
      BEGIN
        multiplier := InvertArray[j,i];
        FOR k := i TO NumberStations + 1 DO
          InvertArray[j, k] := InvertArray[j,k] - multiplier*InvertArray[i,k];
      END(*FOR j...*)
    END(*FOR i...*);

    (*Now solve iteratively backward, from last to first throughput, and
      place in StationThruput array.*)

    FOR i := NumberStations DOWNTO 1 DO
    BEGIN
      temp := InvertArray[i, NumberStations + 1];
      FOR j := i + 1 TO NumberStations DO
        temp := temp - InvertArray[i,j]*StationThruputs[TrafficIndex, j];
      StationThruputs[TrafficIndex, i] := temp/InvertArray[i,i]
    END(*FOR i...*);

    (*Need to perform interchange, if any, of solutions*)

     FOR i := 1 TO TransposeCount DO
     BEGIN
       j := transpose[i,0];
       k := transpose[i,1];
       temp := StationThruputs[TrafficIndex, j];
       StationThruputs[TrafficIndex, j] := StationThruputs[TrafficIndex, k];
       StationThruputs[TrafficIndex, k] := temp
     END(*FOR i...*);
    SolveThruputs := TRUE
  END(*SolveThruputs*);

END(*NetComp*).
```